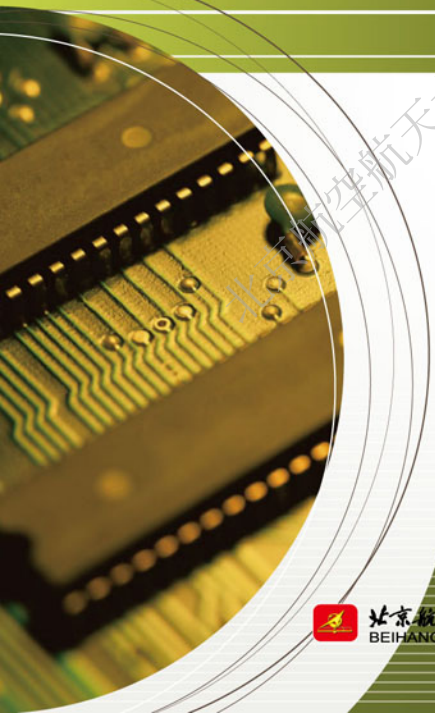
“十二五”高等院校规划教材

51单片机原理与实践

（C语言版）



高卫东 编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

“十二五”高等院校规划教材

51 单片机原理与实践（C 语言版）

高卫东　编著

北京航空航天大学出版社

内 容 简 介

以 51 单片机为载体，以 C 语言为主线，以 Proteus 设计仿真平台为手段，介绍了单片机的内部结构、接口及其应用。以实例需求为知识切入点，充分发挥 C 语言特点，在讲清单片机基本结构的基础上，重点讲解系统扩展及新器件的使用，注重通过原理图设计、源程序编写、软硬件联调来降低学习难度和提高学习质量。

本书注重新技术、新手段、新器件的使用，既可以作为各高等院校单片机课程的教材、课程设计指导，也可作为工程技术人员的进修参考用书以及大学生电子设计竞赛的培训资料。

图书在版编目(CIP)数据

51 单片机原理与实践 ：C 语言版 / 高卫东编著. --
北京 ：北京航空航天大学出版社，2011.1

ISBN 978-7-5124-0293-5

Ⅰ. ①5… Ⅱ. ①高… Ⅲ. ①单片微型计算机－C 语言－程序设计 Ⅳ. ①TP368.1②TP312

中国版本图书馆 CIP 数据核字(2010)第 243287 号

51 单片机原理与实践(C 语言版)

高卫东 编著

责任编辑 冯颖

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191) http://www.buaapress.com.cn

发行部电话：(010)82317024 传真：(010)82328026

读者信箱：emsbook@gmail.com 邮购电话：(010)82316936

印刷有限公司印装 各地书店经销

*

开本：787×1092 1/16 印张：17 字数：435 千字

2011 年 1 月第 1 版 2011 年 1 月第 1 次印刷 印数：5 000 册

ISBN 978-7-5124-0293-5 定价：29.00 元

前　言

单片机技术在电子系统中的应用越来越广泛，已有融于其中之势。目前很多新型电子器件对控制信号要求很高，若不采用单片机或可编程器件则几乎无法使用。单片机技术的应用不仅可以简化电路设计，而且可以大大提高电路设计水平。

单片机知识大体上分为基本结构、编程语言、定时/中断、通信、系统扩展 5 部分。除了基本结构、定时/中断、通信这 3 部分变化不大，其他部分伴随相关技术的发展均有所变化。尤其是具有单片机系统仿真设计功能的 Proteus 软件的出现，使得我们可以在计算机纯软环境中完成单片机电子系统的设计与调试，Proteus 系统库不仅提供了 51、AVR、PIC 等单片机模型，以及大量模拟、数字、光电、机电类元器件模型，还提供了许多虚拟仪器。Keil 环境下开发的程序和 Proteus 设计的仿真电路进行交互式联调运行，解决了长期以来困扰单片机教学的软件和硬件无法很好结合的问题。学会 Proteus 和 Keil 软件的使用，可以极大地降低单片机学习难度，提高学习质量。本书在附录中给出了 Proteus 及 Keil 软件平台的使用方法。

第 3 章“C51 语言程序设计基础”介绍了在 51 单片机中使用的 C 语言。C 语言具有高级语言的功能和低级语言控制硬件的能力。传统的单片机编程采用汇编的居多，用汇编编程不仅需要专门学习汇编语言，而且易与微机原理中的汇编混淆，同时所学汇编语言在其他高档单片机及嵌入式系统中没有用武之地。与此相反，C 语言是电子信息类各专业的通识课程，51 单片机所用的 C 语言（即 C51）和标准 C 语言（即 ANSI C）基本相同，只要掌握其应用方法即可。更重要的是，C 语言是公认的高效简洁、贴近硬件的编程语言，程序可读性好且易于移植，其他高档单片机和嵌入式系统均用 C 语言编程，所以使用 C 语言学习单片机编程，不仅可以充分利用已有知识，而且为进一步学习奠定了基础。同时，Keil 集成环境中的 C51 编译器是 51 单片机最高效、最灵活的开发平台之一，为我们提供了：编译器、汇编器、实时操作系统、项目管理器以及调试器。

一个完整的单片机系统通常包括键盘输入、显示输出、打印输出、数据采集等许多功能模块。这些功能模块一般通过 I/O 端口实现与单片机的数据交换，但是单片机的 I/O 端口有限，且一般用来处理数字信号，从而产生了总线式传输模式。这样，系统扩展的方向也由原来单一的并口应用转到 SPI（Serial Peripheral Interface）、I^2C（Inter IC Bus）、1-Wire 等器件的应用。新颖的外围电路、新器件与新技术为实现优化设计、系统集成提供了技术支持。本书把外部扩展技术的重点放在了以介绍 SPI、I^2C 等串行接口的原理和应用方面，这不仅符合技术的发展方向，同时也更加贴近实际应用。第 7 章介绍了串行总线应用的知识。

单片机的学习不能宥于单片机本身，必须将单片机置于实际系统之中。全书以夯实基础、面向应用，理论与实践、方法与实现紧密结合为主线展开，通过实例来切入知识点，通过具体实际需求的实现来体现单片机技术硬软协同工作的问题。只有让单片机在外围电路环境下工作，让单片机在电子系统中工作，才能真正体会单片机和外围扩展硬件、控制硬件协同工作的

设计思想及实现方法。

考虑到有些读者是从汇编语言转到C语言以及深入研究的需要，本书在附录中给出了相关例题的汇编实现程序。

参加本书编写工作的还有吴祖国、张兴胜、汪小会、毕大庆、王津、海磊等同志，在此一并表示感谢。

本书强调实用性和先进性，目标是将一些新趋势、新技术、新思想融于单片机具体知识的传递过程中，但由于编者水平有限，难免存在不妥之处，敬请广大读者批评指正。

编　者

2011年1月

目　录

第1章 绪 论

本章首先介绍嵌入式系统、单片机(MCU)及其应用系统的基本概念,然后介绍51单片机的基本情况和单片机应用系统的开发过程。

通过一个能使LED(发光二极管)闪烁的应用系统的具体实现过程,使读者对单片机及其应用系统有一个感性认识,并大致了解单片机的基本工作过程,知道单片机应用系统中同时起作用又相辅相成的两个方面:硬件和软件。

1.1 嵌入式系统简介

嵌入式系统是目前电子系统设计最活跃的领域之一,具有广阔的市场前景。经过几十年的发展,嵌入式系统已经在很大程度上改变了人们的生活、工作和娱乐方式。在日常生活中,人们使用各种嵌入式系统(虽然人们未必知道这个名词),如MP3、DVD、DV、PDA、ADSL、手机、打印机等。几乎所有带有一点"智能"的家用电器(如全自动洗衣机、温控冰箱等)都是嵌入式系统。

1.1.1 嵌入式系统概念

根据IEEE(国际电气和电子工程师协会)的定义,嵌入式系统是"控制、监视或者辅助设备、机器和车间运行的设备"(英文为:devices used to control,monitor or assist the operation of equipment,machinery)。这主要是从应用加以定义的,由此可以看出嵌入式系统是软件和硬件的结合体。

国内也有一个普遍的定义:以应用为中心,以计算机技术为基础,软件和硬件可裁剪,适应应用系统对功能、可靠性、成本、体积、功耗等严格要求的"专用计算机系统"。还有一种定义是:嵌入式系统是设计完成复杂功能的硬件和软件并使其紧密耦合在一起的计算机系统。以上两种定义的出发点不一样,前者是从技术的角度来定义的,后者是从系统的角度来定义的。事实上在大多数情况下,嵌入式系统被真正地嵌入,是"系统中的系统",不能没有自己的功能。某些情况下,嵌入式系统在功能上是独立的。例如,网络路由器就是独立的嵌入式系统,它由

特殊的通信处理器、内存、网络端口以及实现路由算法的特殊软件组成。

总之，嵌入式系统是一个外延很广的名词，凡是与产品结合在一起、具有嵌入式特点的控制系统都可以称为嵌入式系统。目前通常把嵌入式系统概念的重点放在“系统”上，把嵌入式系统看作硬件和软件的综合体。硬件一般由高性能的微处理器及其外围的接口电路组成，软件一般由实时操作系统和运行在其上的应用软件构成。

一般而言，嵌入式系统在结构上可以分成 4 部分：处理器、存储器、输入/输出系统和软件（与计算机的结构相同）。

1.1.2 嵌入式处理器

早期，嵌入式系统通常用“通用目的处理器”建造。普通的个人计算机（PC）中的处理器就是通用目的处理器。因其“通用”，功能丰富，所以其结构设计要能满足各方面的要求，导致结构非常复杂，制造成本高昂。

近年来，随着先进的微处理器制造技术的发展，越来越多的嵌入式系统用嵌入式处理器建造，而不再用通用目的处理器。这些嵌入式处理器是为完成特殊应用而设计的“特殊目的”处理器。从硬件上讲，嵌入式系统的核心就是嵌入式处理器。

嵌入式处理器有很多的品种、数量、体系结构。有的嵌入式处理器注重尺寸、能耗和价格，有的更关注性能，还有的嵌入式处理器关注上述全部 4 类需求。

一般来说，嵌入式处理器可以分成以下 4 类。

1. 嵌入式处理器（Micro Processor Unit，MPU）

嵌入式微处理器的基础是通用计算机的 CPU。在应用中，将微处理器装配在专门的电路板上，只保留与嵌入式应用有关的母板功能，以大幅减小系统体积并降低功耗。为了满足嵌入式应用的特殊要求，虽然其功能与标准微处理器基本相同，但在工作温度范围、抗干扰及可靠性等方面作了各种增强。

与前期的工控计算机相比，嵌入式微处理器具有体积小、质量轻、成本低、可靠性高的优点。但是在电路板上必须包括 ROM、RAM、总线接口及各种外设等部件，从而降低了系统的可靠性，技术保密性也较差。

嵌入式微处理器目前主要有 Am186/88、386EX、SC-400、Power PC、6800、MIPS 以及 ARM 系列等。

2. 嵌入式微控制器（Micro Controller Unit，MCU）

嵌入式微控制器一般又称为单片机，顾名思义就是将整个计算机系统集成到一块芯片中。嵌入式微控制器一般以某一种微处理器内核为核心，芯片内部集成了 ROM、RAM、总线、定时/计数器、看门狗、I/O、串行口、PWM（脉宽调制）输出、A/D、D/A、Flash、E^2PROM 等各种嵌入式系统必要的功能和外设。

为适应不同的应用需求，一般一个系列的单片机具有多个衍生产品，每种衍生产品的处理器内核都是一样的，不同的是存储器和外设的配置及封装形式。这样可以使单片机最大限度地与应用需求相匹配，功能不多不少，从而降低功耗和成本。

与嵌入式微处理器相比，微控制器的最大特点是单片化，体积大大减小，功耗和成本下降，

可靠性提高。微控制器是嵌入式系统工业的主流。其片上资源比较丰富，适合于控制，因此称为微控制器。

嵌入式微控制器目前的品种和数量最多，比较有代表性的系列有 MCS－51/96、PIC、MC68 和数目众多的 ARM 芯片等。目前 MCU 占嵌入式系统约 70％的市场份额。

3. 嵌入式 DSP 处理器(Digital Signal Processor，DSP)

在数字滤波、FFT、频谱分析等方面，DSP 算法正在大规模进入嵌入式领域。DSP 应用正在从通用单片机以普通指令实现 DSP 功能，过渡到采用嵌入式 DSP 处理器来实现。DSP 处理器在系统结构和指令方面进行了特殊设计，使其适合于执行 DSP 算法，并专门用于信号处理方面，具有很高的编译效率和指令执行速度。

嵌入式 DSP 处理器比较有代表性的产品是 TI 公司的 TMS320 系列和 Freescale 公司的 DSP56000 系列。

4. 嵌入式片上系统(System on Chip，SoC)

SoC 就是在一个硅片上实现一个更为复杂的系统，其最大特点是实现了软/硬件的无缝结合。各种通用微处理器内核都成为了 SoC 设计公司的标准库，和其他许多嵌入式系统外设一样，成为 VLSI 设计中一种标准的器件，用标准的 VHDL 语言描述。用户只要定义出其整个应用系统，仿真通过后就可将设计图纸交给半导体公司制作样品。除个别无法集成的器件之外，系统大部分均可集成到一块或几块芯片中，应用系统电路板将非常简洁，对减小体积、降低功耗、提高可靠性等都非常有利。

SoC 可以分成通用和专用两类。通用系列包括 Infineon 公司的 TriCore、Freescale 公司的 M-Core、某些 ARM 系列器件以及 Echelon 公司和 Freescale 公司联合研制的 Neuron 芯片等。专用 SoC 一般不为用户所知。

1.2 单片机的基本概念

单片机作为嵌入式系统最典型的代表，其在嵌入式系统产品中占有最大的市场份额，也是广大高校学生学习嵌入式系统的主流。

1.2.1 CPU、微型计算机及微型计算机系统

想要明白什么是单片机，就要从微型计算机的概念入手。

通过“微机原理”相关课程的学习，我们已经了解到一台能够工作的计算机必须具备以下几个基本部分：运算器、存储器、控制器和 I/O 接口。

在微型计算机中，为减小计算机的体积，将运算器和控制器放在一起以一块芯片的形式出现，称为中央处理器 CPU(Central Processing Unit)，也称为微处理器。它是微型计算机最重要的组成部分。

但是，CPU 也仅仅是微型计算机的运算及控制部件，它本身并不能独立工作，不能独立地执行程序。只有将 CPU 和存储器、I/O 接口通过“总线”联系起来，才能构成微型计算机。总线根据传送对象的不同，又可以分成数据总线(DB，Data Bus)、地址总线(AB，Address Bus)以

及控制总线(CB,Control Bus)。

图 1.1 为微型计算机结构示意图。图中 C 表示控制信号,A 表示地址信号,D 表示数据信号。

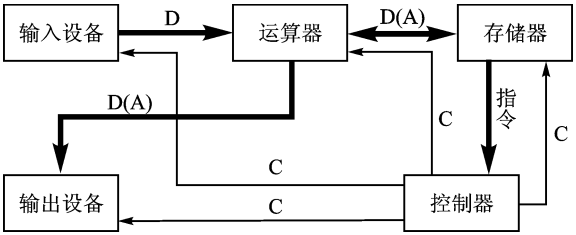


图 1.1 微型计算机结构示意图

有了上面的基本组成部分,微型计算机还不能应用于实际,必须加上输入设备、输出设备等计算机外设和一些计算机的系统软件(如操作系统)及应用软件,才能实现相应的功能。

1.2.2 单片机

在了解了一般计算机的基础知识之后,我们再来看看什么是单片机。

"单片机"是在很多外文资料中提到的 Single Chip MicroComputer 较准确的译法。在早期,Single Chip MicroComputer 准确地体现了单片机的形态和结构。在一般计算机中,它的几个基本组成部分被做成若干块芯片和电路板,通过"主板"相连,是一个"多板、多片"系统。单片机也是一种微型计算机,不同的是单片机将计算机的几个基本组成部分全部做到一块集成电路芯片中,使得这样一块集成电路芯片成为了一台简单的微型计算机,具备了一般计算机的功能,可进行简单的运算和控制。

当然,单片机芯片内部不仅仅只有上面所讲到几个基本部件。单片机主要面向控制,除了进行一些简单运算外,主要完成各种控制功能。在很多控制场合,"计数"和"定时"是少不了的。因此,在单片机这一块芯片中就设置有若干个定时/计数器,免去了用户外接定时芯片的麻烦。

单片机芯片内部还有一个"时钟电路",使得单片机在外接振荡源的作用下,能产生相应频率的时钟信号,使其运算和控制都能有节奏地进行。

另外,单片机芯片内部还有起着"传达室"作用的"中断系统"。当单片机所控制对象的参数到达需要 CPU 加以干预的状态时,就可经此中断系统"通报"给 CPU,使 CPU 根据外部事态的轻重缓急采取适当的应对措施。

单片机芯片内部也有将这些部分连接起来的"总线"。CPU、ROM、RAM、I/O 口、定时/计数器、中断系统等都通过总线连通,一切指令、数据、控制信号都可经内部总线传送。

图 1.2 为 MCS-51 单片机的基本组成示意图。

由此可以看出,单片机虽然只是一个芯片,但一般计算机中的基本部件它都有,因此单片机实际上就是一台简单的微型计算机。

随着单片机的发展,现在的单片机芯片中都着重扩展了各种控制功能。除了集成了定时/计数器外,有的单片机中还集成了如 A/D、D/A 等功能部件(为了加强模拟信号的采集、处

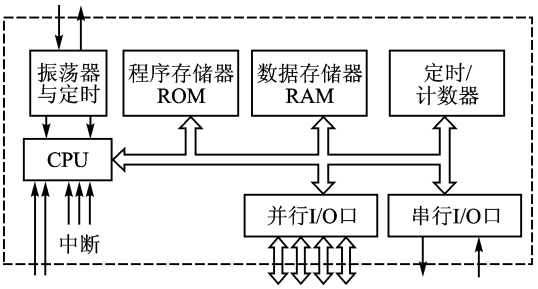


图 1.2　MCS－51 单片机组成示意图

理);有的单片机芯片内部集成了 PWM、PCA(计数器捕获比较逻辑)、高速 I/O 口、WDT(看门狗定时器)等功能部件。现在的单片机在结构上已突破了原来"Single Chip MicroComputer"的传统内涵,最能准确反映其设计思想、具有长远技术眼光的名称应该是"Micro—Controller",即前面所提到的"嵌入式微控制器 MCU"。国外已逐渐统一为"MCU",国内现在仍然保留了"单片机"这个叫法。

1.2.3　单片机应用系统

虽然单片机已经具备一个微型计算机的基本结构和功能,但实质上它也仅仅是一个芯片,而仅有单片机一个芯片还不能完成任何工作。在实际应用中,要让单片机去实现相应的功能,就必须将单片机与被控对象进行电气连接,必须根据需要增加各种扩展接口电路、外部设备和相应软件,从而构成一个"单片机应用系统"。

图 1.3 为单片机应用系统的示意图。

单片机应用系统是以单片机为核心,配以输入、输出、显示、控制等外围电路和相应的控制、驱动软件,能完成一种或多种功能的实用系统。同微型计算机系统一样,单片机应用系统也是由硬件和软件组成的,二者相互依赖,缺一不可。

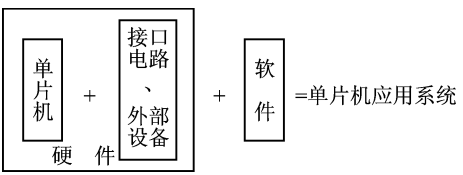


图 1.3　单片机应用系统

由此可见,单片机应用系统的设计人员必须从硬件和软件两个角度来深入了解单片机并能将二者有机地结合起来,才能设计出具有特定功能的应用系统或整机产品。

1.2.4　单片机的发展和应用

1. 单片机的发展

在计算机应用控制领域(如工业控制、汽车、家电等),对控制系统的要求都比较苛刻。例如需要高智能、小体积、低成本、低功耗、强抗干扰能力和高可靠性。不仅传统电气设备无能为力(无智能或低智能),就是一般应用型 PC 机也不能胜任。在这样的背景下,关于单片机的设想才逐渐成形。

单片机就是将计算机的几个基本组成部分集成在单一的芯片上，体积相对较小(一般也就几厘米长，1 cm 左右宽)，能很好地满足对控制系统体积的要求。

很多控制场合并不需要完成复杂的数学计算，因此单片机在生产工艺上进行了简化，降低了制造成本。同时采用大批量生产，成本进一步降低。从目前市场上来看，其价格一般都在几元到几十元之间。

单片机将所有电路都集成在一个芯片上，降低了因为线路连接导致系统失效的可能性，使可靠性提高。

在工艺和设计上采取的措施又使其功耗降低。单片机(尤其是低功耗单片机)功耗比一般的微型计算机的功耗要低得多(mW 级)，满足了对控制系统功耗的要求。

单片机在设计上采用了很多有效的抗干扰措施，使其能够在各种恶劣的环境下都能可靠地工作，如适应温度范围广，在－50～100 ℃范围内都能正常工作。

这些特点都很好地满足了工业控制应用的诸多特殊需求，因此单片机很快进入工业计算机控制的诸多领域，获得了广泛的应用，充分显示了其强大的生命力和广阔的应用前景。

自 1975 年美国德州仪器(TI)公司开发生产出第一台单片机 TMS-1000 以来，单片机已经历了 4 位→8 位→16 位→32 位的发展过程。

最有代表性的是 Intel 公司先后推出了 3 个系列：1976 年推出 MCS－48 系列 8 位单片机；1980 年推出 MCS－51 系列高档增强 8 位单片机；1983 年推出 MCS－96/98 系列 16 位单片机。

其中以 MCS－51 为主流的 8 位机，主要应用在传统电子系统智能化和对象系统的控制领域。目前 8 位单片机的功能还在不断增强，还在不断发展。与 16 位机相比，增强型的 8 位单片机性价比更高。

现在流行的 32 位 ARM 系列，主要应用在网络(如路由器)、通信(如手机)和多媒体技术等领域。

2. 单片机的应用

单片机的价值在于实现计算机的“在线控制”。所谓“在线控制”，就是以计算机代替控制系统中常规的模拟或数字电路，使计算机成为控制系统的一部分。由于计算机身处其中，因此对它有体积小、功耗低、价格低、控制功能强等要求，单片机正好具备了这样的优点：

- 体积小，成本低，运用灵活，能方便地组成各种智能化的控制设备和仪器。
- 面向控制，能有针对性地解决各类控制任务，能获得最佳性价比。单片机是面向控制而设计的，在内部结构上采取了很多有利于控制的一些结构，如定时器、特殊功能寄存器、位处理器等，因此能非常有针对性地解决各类控制任务。
- 抗干扰能力强，适应温度范围广，在各种恶劣的环境下都能可靠地工作。
- 功耗低，在供电不方便的时候也能正常运行。

当然，一般计算机也能在控制领域发挥作用，比如单片机应用系统中的软件设计就要在计算机上进行。但这是计算机在控制系统中的“离线应用”，就是利用通用计算机完成控制系统的分析、设计、仿真、建模等工作。这对单片机来说，自然是无法胜任的。

单片机的应用范围十分广泛，目前单片机已成为航空航天、机械电子等工业测控领域中最为理想的控制计算机。

下面列举单片机的一些典型应用领域：

- 日常生活：如电梯、商场中的电子秤、条码阅读机、IC 卡刷卡机、银行里的电子点钞机、高级玩具、各种智能型家用电器、出租车的计价器等；
- 计算机外设及办公自动化：如键盘、打印机、绘图仪、磁盘驱动器、传真机、复印机等；
- 智能化仪器仪表：如智能仪器、医疗检测器械、数字示波器等；
- 工业控制：如数控机床、温度控制、工业机器人、智能传感器等；
- 汽车工业：如点火控制、变速控制、排气控制、转向控制；
- 导航与控制：如导弹控制、智能武器装置、航天导航系统等。

通过上面列举的部分应用，再对比早期的一些电子产品，可以看出：单片机的出现和应用，正从根本上改变着控制系统的设计思想和设计方法。现在有些技术人员或其他业余电子开发者研制出来的某些产品，不是电路太复杂不实用，就是功能太简单、缺乏智能性且极易被仿制。究其原因，就是因为他们采用的是传统的控制系统设计思想，在产品中没有使用单片机或其他可编程逻辑器件，所有功能都采用模拟和数字电路实现。

现代控制系统设计思想是将单片机或其他可编程器件与传统模拟、数字技术相结合，原来需要通过模拟电路、脉冲电路、组合逻辑实现的控制功能，现在相当大的一部分都可以利用各种单片机通过软件方法予以实现。这种用软件代替硬件并能提高系统性能的控制技术，我们称之为“微控制技术”。控制系统的设计方法正在演变成软件和硬件相结合的方法，许多电路设计问题转化为程序设计问题，即人们所说的“软件就是硬件”。这样做的优势显而易见：

- 电路相对简单很多，使用器件数量减少，故障概率低；
- 产品不容易被仿制；
- 有利于产品升级换代，形成“智能型”、“数字化”、“自动化”产品，如智能温控冰箱、全自动洗衣机、数字电视等；
- 产品功能扩展方便，参数变化灵活。

1.3 51 系列单片机简介

1.3.1 8 位单片机是嵌入式系统低端应用的主流

嵌入式系统的高、低端应用可模糊地界定为：低端用于对象系统的控制领域，高端用于具有“海量”数据处理的网络、通信和多媒体技术等领域。

在低端应用方面，8 位单片机是满足绝大多数的对象控制要求的最佳选择。在实际工作中并不是都要求计算机有很高的性能，工业控制更是如此。在绝大多数场合，传统电子系统智能化、自动化的要求并不是很复杂，8 位单片机完全可以解决问题。如采用高性能的 16 位机、32 位机，不仅是一种资源浪费，而且延长了开发周期，提高了成本。

当然，8 位机无法满足网络、通信和多媒体技术等领域高端应用的要求，这时往往需要 32 位机来解决，如 ARM 芯片。

1.3.2 51 系列 8 位单片机基本情况介绍

尽管 8 位单片机种类很多，但无论是从全国范围还是从世界范围来看，51 系列都是使用最广泛、影响最深远的，许多公司都推出了兼容系列单片机。51 系列单片机实际上已经成为公认的 8 位单片机的标准，一直占有最大的市场份额。

从目前的情况看，MCS－51 内核系列兼容的单片机仍是嵌入式系统低端应用的主流产品，各高校及专业学校仍以 MCS－51 单片机作为代表进行嵌入式系统理论基础的学习。

MCS－51 系列单片机是由美国 Intel 公司开发、研制、生产，并于 1980 年推出的产品。MCS－51 系列包括了很多型号的芯片，如 8051、8031 等。

其中 8051 是 51 系列中最早最典型的产品，它最能体现单片机“Single Chip Microcomputer”的基本结构。51 系列的其他单片机都是在 8051 的基础上进行功能和结构上的增、减、改变而来的。因此，以前的很多文献习惯于用 8051 来代表 MCS－51 系列单片机。但由于 8051 芯片使用起来不方便，功耗也较高，在实际使用方面早已经被淘汰，代之以其兼容的 8031、89C51 等芯片。

表 1.1 给出了 51 系列单片机的基本情况。

表 1.1　MCS－51 系列单片机的特性

特　性	51 子系列 (8031*、8051、8751) (80C31、80C51、87C51) (89C51、89S51)	52 子系列 (8032*、8052、8752) (80C32、80C52、87C52) (89C52、89S52)
片内 ROM	4 KB	8 KB
片内 RAM	128 B	256 B
寻址范围	64 KB＋64 KB	64 KB＋64 KB
定时/计数器	2×16 位	3×16 位
并行 I/O	4×8	4×8
串行 I/O	1 个	1 个
中断源	5 个	6 个

注：*　在 8031、8032 单片机中没有片内 ROM。

1. 51 子系列和 52 子系列

MCS－51 系列又可分为 51 和 52 两个子系列，如 8031、8032。其中 51 子系列为基本型，52 子系列为强化型。以下如果不特别说明，MCS－51 就是指 51 子系列。

52 子系列功能强化主要体现在以下几个具体方面：

- 片内 ROM 容量从 4 KB 增加到 8 KB；
- 片内 RAM 容量从 128 B 增加到 256 B；
- 定时/计数器数量从 2 个增加到 3 个；
- 中断源数量从 5 个增加到 6 个。

2. 51 系列单片机半导体工艺

51 系列单片机采用两种半导体工艺：一种是 HMOS 工艺；另一种是 CHMOS 工艺。在单片机芯片型号中带有字母“C”的，指的是采用 CHMOS 工艺。除保持了 HMOS 高速度和高密度的特点之外，CHMOS 还具有低功耗的特点。例如：8051 的功耗为 630 mW，而 80C51 的功耗只有 120 mW。

在便携式、手提式和野外作业仪器设备上，低功耗是非常有意义的。

1.3.3 两种主流的 51 单片机芯片

近年来，Intel 公司将 MCS－51 的核心技术授权给了 ATMEL、NXP 等公司，现在很多公司都能开发、生产 51 核心的单片机，当然功能或多或少都有些改变，以满足不同场合的需求。

下面介绍两种比较流行的 51 系列单片机芯片。

1. AT89C51

AT89C51 是这几年我国非常流行的单片机，由 ATMEL 公司开发生产，在 8051、8751 的基础上增强了许多特性。如时钟频率更高，运行速度更快；采用 CHMOS 工艺，功耗更低；工作电压范围更大；其最大的提高还是内部程序存储器由原来的 ROM 或 EPROM，转变成 Flash 存储器，使用更方便，寿命更长，可以反复擦写 1000 次以上。

2. AT89S51

与 PIC 单片机相比，AT89C51 最大的缺陷在于不支持 ISP(在线编程)功能。AT89S51 就是在这样的背景下取代 AT89C51 的。

AT89S51 向下完全兼容 51 系列的所有产品。相对于 AT89C51，AT89S51 单片机在结构和功能上有了一些新变化。最典型的就是支持 ISP 在线编程功能、支持程序存储器串行写入方式、写入电压更低、反复烧写次数更多、工作频率更高、电源适应范围更宽、抗干扰性更强、加密功能更强、支持低功耗模式。另外，AT89S51 在结构上还设计了双数据指针(Dual Data Pointer)，设置了电源关闭标志(Power Off Flag)。

现在 ATMEL 已经停产 AT89C51，因此在市场上见到的 AT89C51 实际都是 ATMEL 前期生产的库存芯片。

1.4 单片机应用系统的开发过程

单片机虽然可方便地组成各种不同的应用系统，但由于其自身无调试能力，因此必须配备一定的开发工具来开发应用软件和对硬件电路进行诊断。比如单片机应用系统建立以后，电路正确与否，需要一定的工具来测试；程序是否有误，需要一定的手段来调试；将程序装入系统内部也需要通过一定的设备来实现。这种工具就是单片机开发系统。

1.4.1 开发系统的作用

开发系统的功能应该有以下几个功能：

- 系统硬件电路的诊断和检查；
- 程序的输入和修改；
- 程序的调试、运行，具有单步运行、断点运行等各种调试功能；
- 能将调试好的程序代码写入到程序存储器中。

1.4.2 开发系统的组成

除了进行硬件电路检查所需要的一些常用设备(如通用的示波器、万用表等)外，单片机的开发系统还应该包括下面的几个部分。

1. 通用计算机

利用它编写用户程序和进行用户程序的调试。对计算机的要求不高，只要能流畅地运行一些“编辑软件”和下面要讲的“仿真软件”即可。

2. 仿真器

仿真器是开发系统中最关键的部件，它是将应用系统和主机连接起来的桥梁。在应用程序尚未调试好时，可借用仿真器所提供的 CPU、存储器等资源来调试程序。

3. 仿真软件

提供汇编、反汇编、跟踪执行、单步执行等各种调试手段，使用户可方便地查找程序中的各种语法和功能错误。

4. 编程器

当系统调试完毕，确认软、硬件无故障时，程序就应该脱离仿真器而独立运行。这时需要将用户应用系统程序的机器代码烧写到单片机的程序存储器中。编程器就是完成这种任务的专用设备。

1.4.3 仿 真

在实际应用开发过程中，程序员不能保证万无一失，如果将错误的机器代码写入存储器后，发现不对了再擦除、烧写，再发现不对了再擦除、烧写，这样肯定很麻烦。为了节约成本，缩短开发时间，应该在确保程序完全正确、合理以后，再将机器代码写入程序存储器。在此之前，应该先模拟“演练”一下，也就是先进行“仿真”。仿真的主要目的是进行软件调试(当然也能进行一些硬件排错)。

仿真需要借助开发系统中的仿真器。仿真器本身也是一个特殊的单片机应用系统，与所要开发的单片机应用系统具有相同系列的单片机芯片。一块用户开发、制作的单片机应用电路板包括单片机部分和为实现设计功能而设计的外围电路部分。由于单片机部分自身没有调试能力，无法验证系统的正确性，这时就用仿真器来代替应用电路板中的单片机部分，模拟出目标系统的 CPU、ROM、RAM 和 I/O 口，使仿真时目标系统的运行环境和脱机时运行的环境完全“逼真”。

图 1.4 所示为单片机的仿真示意图。

具体操作方法是：在单片机应用系统电路板制作完成后，电路板上单片机芯片和外接的程序存储器芯片（如果有的话）暂时不插，先用仿真器提供的仿真头代替。这样一方面可将仿真器中的单片机暂时借给应用系统用作 CPU，另一方面也将仿真器中的 RAM 也借给应用系统用作程序存储器（否则应用程序没地方存放），利用这些模拟出的环境进行调试。

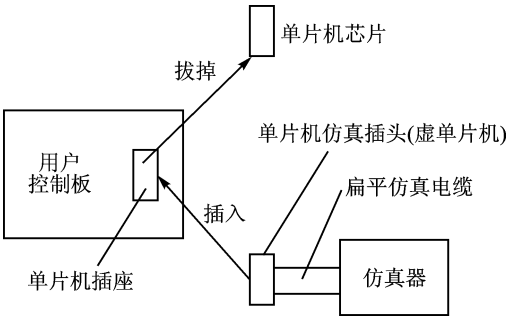


图 1.4　仿真示意图

仿真是单片机开发过程中非常重要的一个环节，一般产品开发过程中都要进行仿真。待程序仿真运行无误、一切功能都实现以后，就可以将调试好的程序机器代码烧写到单片机的程序存储器中，然后将单片机芯片插到用户电路板上、接通电源，一般情况下系统就可以脱机运行了。

1.4.4　单片机应用系统的开发过程

单片机应用系统从提出任务到正式投入运行的过程，称为单片机的开发。一般包括下面几个步骤：

① 确定任务。就是确定系统的功能和技术指标。

② 总体设计。根据应用系统提出的各项技术性能指标，制定性价比高的方案。其中对软件和硬件进行分工是一个最重要的环节。

③ 硬件设计。指应用系统的电路设计，根据设计要求设计、制作、测试好电路板，排除系统中明显的硬件故障。

④ 软件设计。根据应用系统的功能要求编写程序。

— 首先要确定一些常数、地址。这些常数、地址在设计硬件阶段已被直接或间接地确定下来。如当某器件的连线设计好后，其地址也就确定了；当器件的功能确定下来后，其控制字也就相应确定了。

— 然后在 PC 机上编写用户源程序。

— 最后运行仿真软件，对源程序文件汇编（或编译）、查错，形成机器代码。

⑤ 系统调试。通常是硬件和软件结合起来进行调试。除极简单的程序外，一般都需要借助仿真器对软件进行调试，找到并修正其中可能存在的功能错误，直到程序运行正确为止。

⑥ 将调试好的程序固化到程序存储器中。在使用仿真软件对源程序汇编时，会生成扩展名为 Hex 的目标文件（机器代码文件，编程器都能识别这种格式的文件）。可用编程器将程序代码写入到单片机内部的程序存储器或外接的存储器芯片中。

1.5　任务 1：信号灯控制实战

下面借助一个非常简单的单片机应用系统的开发过程，让大家对单片机应用系统的开发过程有个感性认识。因功能简单，这里不借助仿真器。

1.5.1 实现功能要求

以 MCS－51 系列单片机的 AT89C51 芯片为核心，参照图 1.5、图 1.6，进行相应的电路连接。并编写相应程序。当系统运行时，图 1.6 中连接的 8 个发光二极管同时闪烁，闪烁的速度可通过程序设置。

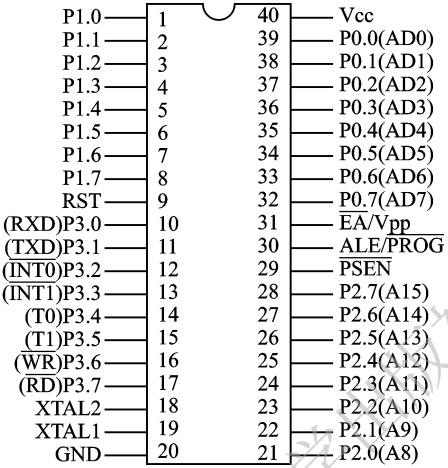


图 1.5　MCS－51 单片机引脚分布图(40 引脚)

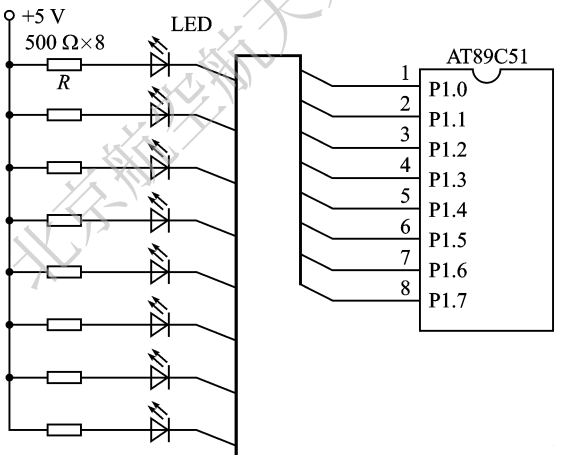


图 1.6　任务 1 电路图

1.5.2 硬件电路连接

1. MCS－51 单片机的引脚分布

用单片机来构造一个单片机应用系统，需要将单片机和外部设备进行电气连接。那么就必须熟悉和了解单片机的各个引脚到底有什么功能。

我们要使用的是前面提到的 MCS－51 系列单片机的 AT89C51 芯片。该单片机芯片采

用 40 个引脚的 DIP 封装(双列直插式,Dual In-Line Package),如图 1.5 所示。对于 8031/32、8051/52、89C51/52、89S51/52 来说,引脚分布是相同的。

各引脚的具体作用在后面用到时再详细介绍。在做具体系统时只需要使用其中的一部分引脚,一般不会将它的 40 个引脚全部用完。

2. 构造 MCS－51 单片机的最小系统

下面结合图 1.5 的引脚图,根据图 1.7 找出相应的引脚位置,并进行相应的线路连接。首先可以找一块面包板、一些细导线,然后在面包板上进行线路连接。

① 电源:40 引脚 Vcc 接＋5 V 电源,20 引脚 Vss 接地。

② 振荡电路:准备频率为 6 MHz 晶体振荡器、2 个 20～30 pF 电容(瓷片电容,不需要分正、负极),按图 1.7 的接法连接在单片机的 18、19 脚上。

③ 复位电路:准备 22 μF 左右的电容(电解电容,有正、负极)和 1 kΩ 左右的电阻,按图 1.7 连好。

④ $\overline{\text{EA}}$引脚:$\overline{\text{EA}}$引脚接到＋5 V。

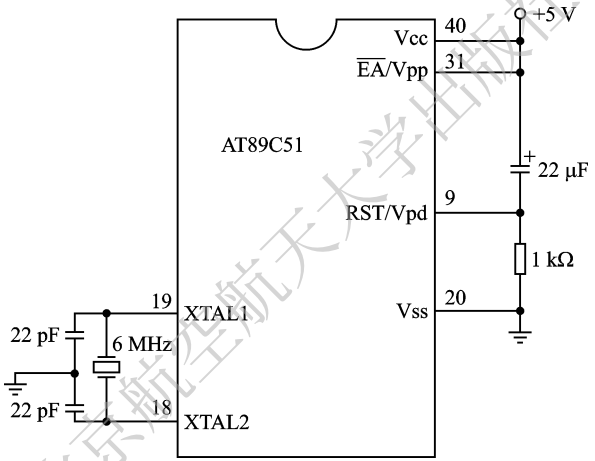


图 1.7　51 单片机的最小系统

到此为止,实际上已经建立起一个单片机最小系统。关于单片机最小系统,这里先给出一个简单易懂的概念:单片机最小系统都是由组成单片机应用系统所必需的一些部件和电路构成的。无论应用系统要完成什么功能,这些部件和电路都是必须的。一个单片机应用系统至少要有:单片机芯片、保证单片机正常工作必须要有电源、产生时钟信号的晶体振荡器,还需要有能使单片机复位的电路等。

3. 连接系统其他部分电路

连接好单片机最小系统后,再在其基础上连接完成预定功能所需要的电路部分:将 8 个 LED(区分正、负极)按图 1.6 与 51 单片机的 P1.0～P1.7 引脚相连。

图 1.7 中限流电阻 R 的作用是保护 LED,使流过 LED 的电流在最大允许电流范围内。一般而言,发光二极管最大允许电流约为 10 mA,因此该电阻的阻值可选为几百欧姆(Ω)。阻值太小,会使通过电流过大导致 LED 烧坏;阻值太大,又会使通过电流过小而导致 LED 亮度不够或不能点亮。

1.5.3 任务分析与实现

连接好线路之后通电，如果振荡电路和复位电路正常的话，单片机就开始工作了。但单片机的工作过程实际上是CPU执行程序的过程，现在还没有编写使LED闪烁的程序，所以此时LED自然不会闪烁。

让8个LED闪烁看似简单，但对于初学者来说，实现起来也是不容易的。为了便于读者理解，可将这个任务分成下面几个步骤实现。

1. 点亮一个LED

先分析如何点亮一个LED(如连接在第1引脚上的LED1)。

按照图1.6的接法，LED1的正极通过限流电阻R接+5 V，负极是接在第1引脚的。根据二极管的单向导电性，当第1引脚电平为高时，LED1不会亮，只有第1引脚电平为低时，LED1才会因导通而发光。因此通过程序控制第1引脚电平的高低，就可以控制LED1的亮灭。

既然要程序控制第1引脚的电平，就得给它起个名字。设计MCS-51单片机的Intel公司已经为它起好了名字，就是P1.0。这是规定好的，不能更改。单片机的指令系统中有一条让P1.0引脚输出高电平的指令，就是“SETB P1.0”，也有一个让P1.0引脚输出低电平的指令，就是“CLR P1.0”。

但要让单片机执行这些指令，还得经过下面两步：

① 单片机不能识别“SETB P1.0”、“CLR P1.0”这种形式的指令，它只能识别电平的高低，也就是0、1这两个二进制数字所代表的状态。因此必须把“SETB P1.0”、“CLR P1.0”形式的指令翻译成这种单片机能识别的形式。这里把“SETB P1.0”变为翻译成“D2 90”，把“CLR P1.0”翻译成“C2 90”。为什么分别是这两个数据，我们现在不去研究。现在先记住这是指令的操作码和操作数，每一条指令都会对应不同的操作码和操作数。

② 将这种数据写入单片机的内部，单片机才能执行。这要借助于“编程器”。将AT89C51单片机芯片插在编程器上，并将编程器与电脑连好，运行编程器的软件(具体怎么操作参见相应的说明书)。

附录C中给出了一个具体的编程器的使用方法，有兴趣的读者可以参考。

先在编缉区内写入“D2 90”，烧写好后，取出单片机将其插入做好的电路板上，接通电源。LED亮不亮？不亮。因为我们写进去的指令“D2 90”(也就是“SETB P1.0”)让P1.0输出高电平，LED1当然不亮。

再从电路板上拔下单片机，重新放回到编程器上，将编缉区的内容改为“C2 90”(也就是“CLR P1.0”)，烧写好后，取出单片机将其插进电路板，接电，这时LED1亮了。

2. 使一个LED闪烁

下面再来分析如何使一个LED不断地闪烁。

其实很简单，就是要它亮一段时间，再灭一段时间，也就是说让P1.0间断地输出高电平和低电平。请考虑将两条指令的代码“C2 90”和“D2 90”都写入单片机的方法是否可行：

```
CLR  P1.0     ;(1)C2 90
SETB P1.0     ;(2)D2 90
```

这样做不行，因为这里存在两个问题：

首先，单片机执行一条指令的时间很短，执行完“CLR P1.0”后，LED1 是该亮了，但在极短时间（μs 级）后，单片机又执行了“SETB P1.0”指令，LED1 又灭了。中间间隔的时间非常短，人的肉眼根本分辨不出 LED 是否亮过。

其次，在执行完“SETB P1.0”后，不能回去执行“CLR P1.0”指令，也就没有机会再让 LED1 亮，自然就产生不了闪烁的效果。

为解决这两个问题，可以做如下设想：

第一，在执行完“CLR P1.0”后，过一段时间再去执行“SETB P1.0”。这样就可以分辨出 LED1 曾经亮过（时间长短无所谓，只要人的眼睛能分辨得出就行）。

第二，在执行完“SETB P1.0”指令后，过一段时间再回头去执行“CLR P1.0”指令，并不断地“循环”。

将两者结合起来，这样就可以使一个 LED 不断地闪烁。

以下先给出程序（后面括号中的数字是为了便于讲解而写的，实际不用输入）：

```
        ORG     0000H
LOOP:   SETB    P1.0            ;(1)D2 90
        LCALL   DELAY           ;(2)12 00 0C
        CLR     P1.0            ;(3)C2 90
        LCALL   DELAY           ;(4)12 00 0C
        SJMP    LOOP            ;(5)80 F4
DELAY:  MOV     R0,#0FFH        ;(6)78 FF
D1:     MOV     R1,#0FFH        ;(7)79 FF
D2:     DJNZ    R1,D2           ;(8)D9 FE
        DJNZ    R0,D1           ;(9)D8 FA
        RET                     ;(10)22
        END
```

按上面的设想分析一下前面的 5 条指令：

第 1 条和第 3 条不再介绍。

第 2 条和第 4 条一样，功能是调用一段子程序，目的在于延时。从标号为“DELAY”的这一行到“RET”这一行中的所有程序，就是调用的“延时”子程序，大概延时零点几秒（s）。至于具体的时间，以后再学习如何计算。

第 5 条指令“SJMP”中有“JMP”，表示转移。后面跟的是 LOOP，在第 1 条指令的前面也有一个 LOOP。也就是说程序要回去执行第 1 条指令。

这样一来，就可得知程序的执行流程：

执行第 1 条指令，让 LED1 灭。执行第 2 条指令延时一段时间后，执行第 3 条指令使 LED1 亮。执行第 4 条指令延时一段时间后，执行第 5 条指令使程序转移到第 1 条开始执行。如此周而复始，LED1 就在不断地亮、灭闪烁了。

每条指令最后的数据是上述每条指令所对应的机器码，同样也要借助编程器将这些机器码写到单片机内部。在编缉区内依次写入所有指令的机器码“D2 90 12 00 0C …22”，操作方法和前面的一样。烧写好后，取出单片机将其插入做好的电路板，接通电源。可以看到，接在 P1.0 上的 LED1 在不断闪烁。

3. 使 8 个 LED 同时闪烁

若让其余的 7 根线 P1.1～P1.7 也完成和 P1.0 同样的动作，就可以让 8 个 LED 同时闪烁。将程序改变一下：

```
        ORG     0000H
LOOP:   MOV     P1,#0FFH            ;(1)75 90 FF
        LCALL   DELAY               ;(2)12 00 0E
        MOV     P1,#00H             ;(3)75 90 00
        LCALL   DELAY               ;(4)12 00 0E
        SJMP    LOOP                ;(5)80 F2
DELAY:  MOV     R0,#0FFH            ;(6)78 FF
D1:     MOV     R1,#0FFH            ;(7)79 FF
D2:     DJNZ    R1,D2               ;(8)D9 FE
        DJNZ    R0,D1               ;(9)D8 FA
        RET                         ;(10)22
        END
```

这里是将原来程序中的“SETB P1.0”和“CLR P1.0”指令分别变为“MOV P1,#0FFH”和“MOV P1,#00H”指令，意思是让 P1.0～P1.7 的 8 位同时输出高电平和低电平。**注意：**指令变了，相应的机器代码也要变。

同样，用编程器将程序的机器码写到单片机内部。在编缉区内依次写入所有指令的机器代码“75 90 FF 12 00 0C…22”，操作方法和前面的一样。烧写好后，取出单片机将其插入做好的电路板，接通电源，可以看到 8 个 LED 在同时闪烁。

4. 改变 LED 的闪烁速度

如将程序再进行修改，将“MOV R0,#0FFH”指令中的 0FFH 分别改为 80H、40H、20H、10H、08H 等(机器代码也要进行相应的变化)，烧写好后，观察程序的运行结果。我们发现，LED 的闪烁速度越来越快。也就是我们通过改变程序中的参数改变了延时时间。如果将该值减小到某种程度，我们会看到发光二极管就一直在亮(亮度不是很高)，不再闪烁了。其实这是一个错觉，这是因为该值太小，延时时间已经足够短而导致亮、灭的变化速度太快，人的肉眼已经分辨不出。

1.5.4 小 结

以上我们使用了 AT89C51 这样一个具体的单片机芯片来控制信号灯的闪烁，相信大家对单片机及其工作、开发过程都有了一个感性认识，也有了一定的体会：

① 清楚地认识到单片机和单片机应用系统这两个概念的区别。AT89C51 芯片只是一个单片机，要想在它的基础上形成一个能完成如“让 LED 闪烁”这样具体功能的应用系统：在硬件上，要在它的周围搭建好电源、复位电路、振荡电路，还要在 P1 口上连接好 LED；在软件上，还要编写相应的程序。

② 从任务实现的过程中，发现了这样的现象：硬件电路的连线没有做任何改变，但改变了写入单片机中的内容(也就是程序)，就可以改变 LED 显示的效果。因此即使硬件电路一

样，编写的程序不同，所达到的功能肯定也不同。

③ 如果不用单片机，当然也可以完成任务。但这样一方面势必要设计出比图 1.6 复杂得多的电路；另一方面类似改变“闪烁速度”的问题就不好解决。

④ 通过上述功能的实现，对单片机应用系统的开发过程有了一个认识：从分析任务开始，接着进行硬件电路设计，然后进行软件设计与调试，最后将调试好的程序机器代码写入单片机，系统运行。

1.6 任务 2：信号灯控制实战之 Proteus 仿真

1.6.1 Proteus 和 Keil 软件

在任务 1 的完成过程中，只是用到了“编程器”这样的工具。这是因任务比较简单，程序很短，操作起来可以保证不会出现什么错误，所以也就没用到“仿真器”。但如果任务复杂一点（程序长一点），将程序翻译成机器码、将机器码写入编辑区等操作过程中都很容易出错，这时就需要用到仿真软件和仿真器（硬件仿真）。

在进行硬件仿真之前必须先设计好硬件电路的物理原型，需要购买各种元件、焊好电路板或在面包板上搭好电路，还需要有万用表等相应的测试工具。这些对于初学者来说，是一个比较大的投资。而对于高校学生来说，虽然大多数学校建立了的单片机实验室，但大都采用硬件仿真器搭配目标实验板（或整合成一个实验箱）的方式。这种模式使得学生的实践活动只能局限于实验系统所给出的硬件，由于其中大部分硬件电路已经连接好，学生也只能是根据实验项目进行很少部分的线路连接。即使学生在实验室中按照要求做完了规定的实验，也只能说学生对单片机的知识点有了较深入的理解，有了一定的程序编写经验，但在硬件上根本找不到“设计”的感觉。这也说明学生并没有真正具备了单片机应用系统的研制、设计能力。

Proteus 软件是一款功能强大的电路设计分析软件，因为能对嵌入式系统进行软硬件协同设计与仿真，成为嵌入式系统领域最先进的开发工具软件。有了 Proteus 软件，学生可以根据自己的任务要求、爱好，设计出自己满意的“作品”，激发了学生动手自己做“产品”的积极性。

Keil 是一款用于单片机汇编语言和 C 语言编程的软件平台，是通用的单片机软件编写、调试软件环境。我们可以将 Keil 和 Proteus 软件结合起来使用，进行软件设计与仿真。当然在利用 Proteus 软件进行单片机的软件仿真时，仅仅是使用了其部分功能（如使用了其中的 ISIS 智能原理图输入系统来绘制电路原理图）。

关于 Proteus 和 Keil 软件的基本介绍及操作方法请读者参考附录 B。

下面仍然以任务 1 中“完成使 8 个 LED 闪烁”的任务为例，介绍一下用 Proteus 和 Keil 进行仿真的具体过程。

1.6.2 绘制 Proteus 电原理图

在运行 Proteus ISIS 的执行程序后，进入 Proteus ISIS 编辑环境。按照图 1.8 绘制 Proteus 电原理图。

在 Proteus 中绘制电原理图时，最小系统中的晶振电路、复位电路等都可以不画，以节约使用者的时间。

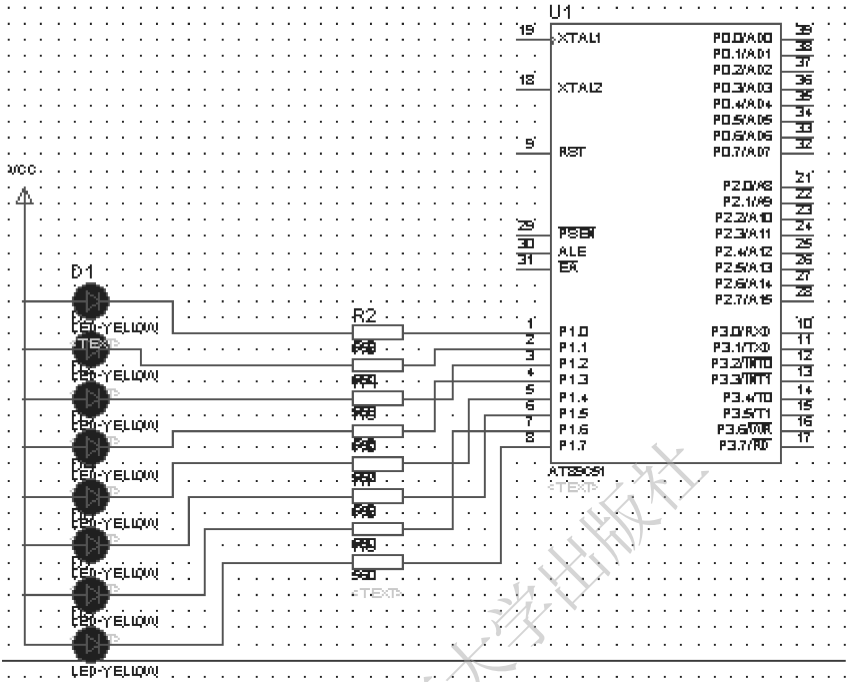


图 1.8　任务 2 之 Proteus 电原理图

1.6.3　编写、汇编、运行程序

步骤如下：

① 在 Keil 中新建并保存项目(如任务 2)，选择 CPU 类型为“AT89C51”。

② 在 Keil 新建文档，输入下面的程序，并保存为“renwu2.asm”。

输入 1.5 节任务 1 中“使 8 个 LED 闪烁”的源程序。这时大家可以发现在输入汇编源代码时，汇编语言的关键字会以不同的颜色显示，这样可以减少语法错误。

```
        ORG     0000H
LOOP:   MOV     P1,#0FFH
        LCALL   DELAY
        MOV     P1,#00H
        LCALL   DELAY
        SJMP    LOOP
DELAY:  MOV     R0,#0FFH
D1:     MOV     R1,#0FFH
D2:     DJNZ    R1,D2
        DJNZ    R0,D1
        RET
        END
```

③ 在 Keil 中对源程序 renwu2.asm 进行汇编，得到 renwu2.hex 文件。

④ 给 CPU 载入程序。在如图 1.9 所示的 CPU 属性对话框中的 Program File 栏中选中编译好的 renwu2.hex 文件，在 Clock Frequency 栏中输入“6 MHz”，然后单击 OK 按钮。

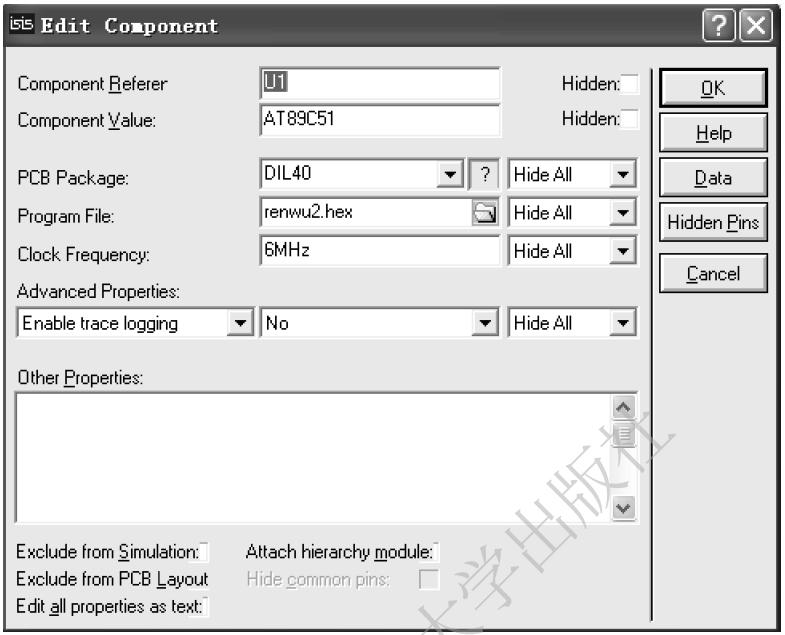
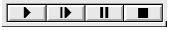
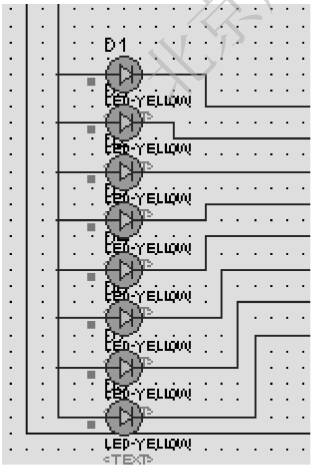
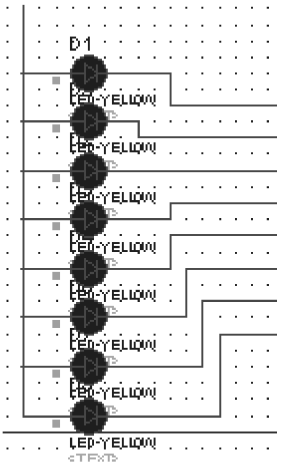


图 1.9　CPU 属性对话框为 CPU 载入程序

⑤ 在 Proteus 中运行程序。单击 Proteus ISIS 编辑界面中  的左下角的运行键  ，使它处于运行状态。这时大家可以看到如图 1.10 所示“8 个 LED 同时闪烁”的情形。



(a) 同时亮



(b) 同时灭

图 1.10　8 个 LED 同时亮和同时灭

1.6.4 改变闪烁速度

在图 1.9 中的 Clock Frequency 栏中输入“12 MHz”，然后单击 OK 按钮。或者将程序中 R0 的值改变一下，如改为 80H，然后进行重新编译（**注意：**程序改变了，一定要重新编译，否则 *.hex 文件得不到修改）。再次运行程序，观察闪烁速度。

第 2 章

MCS－51 单片机的组成和结构分析

通过第 1 章相关内容的介绍，读者已经对单片机的基本概念、单片机由哪些部件组成有了基本的了解。相信大家已经对用单片机去控制一个对象，使其能够实现一定的功能有了一个初步的认识。大家可能也还有些问题不太明白，如：用编程器把程序代码写进单片机内部，这些程序代码存放在什么地方？程序中的一些符号（如 P1.0、P1、R0、R1 等）都表示什么含义？怎样使用它们？构造最小系统时为什么要那样连接线路？程序中的延时程序能延长多长时间？

本章将详细分析 MCS－51 单片机内部存储器的结构、引脚信号、时钟与复位电路以及单片机的 4 个并行 I/O 口。

MCS－51 单片机内部还有其他的功能部件，如定时/计数器、中断系统、串行通信口。由于在讲述它们时需要对其进行编程，因此定时器、串行口、中断的硬件原理部分见后面的相关章节。

2.1 MCS－51 单片机的存储器结构

完成 1.5 节任务 1 的过程中，先用编程器把程序的机器码写进单片机内部，然后取下单片机，插到电路板上，接通电源，然后单片机执行这些指令。那么这些程序机器代码是存放在什么地方的呢？

能够存放程序或数据的器件称为“存储器”，是计算机系统的基本组成部分之一。与一般计算机不同，单片机中的存储器不是独立的，它是和 CPU、I/O 接口等部件集成在一块芯片上的。

2.1.1 MCS－51 单片机的存储器空间

1. 内部程序存储器

在 1.5 节任务 1 的实现过程中，从用编程器把程序机器代码写进存储器，到将单片机插到电路板上、通电，这中间有一个掉电的过程。可见单片机中用来存放程序代码的存储器（本书中称为程序存储器）在掉电的情况下依然可以保存程序代码。

在各种存储器中，如 ROM、EPROM、E^2PROM、Flash ROM，都具有这种“只读”、“非易失性”的功能。说到“只读”，并不是一定不能写（不然也不能用编程器将程序代码写进去），只不过“可写”是在特殊条件下（如高电压）、用特定设备（如编程器）才能对 ROM 进行写操作。在单片机正常工作时，只能从中读数据，不能把数据写进去，因此通常还是把它称为 ROM（Read Only Memory）。

2. 内部数据存储器

单片机应用系统中仅有程序存储器是不够的。在程序运行过程中，肯定要经过许多中间过程，产生许多中间结果，中间结果当然也需要存放在存储器中。但中间结果随时可能变化，而程序存储器在正常工作时是不可写的，因此不能将这些中间结果存放在程序存储器中。所以在单片机中还必须有一个可读可写、用来存放中间结果的存储器，称为数据存储器，由可读可写的 RAM（Random Access Memory）构成。

综上所述： *在 MCS－51 单片机内部集成了具有一定容量的程序存储器和数据存储器，称之为内部程序存储器和内部数据存储器。其具体容量是多大，不同芯片可能会有所不同。*

1.5 节任务 1 中使用到的 AT89C51 单片机芯片内部就有 4 KB 的 Flash ROM 形式的程序存储器，程序代码“75 90 FF 12 00 0C…22”就存放在其中。在该程序执行过程中，R0、R1 的内容都是在不断变化的（从 0FFH 一直减到 00H，然后反复），是一些中间结果。AT89C51 芯片内部有 256 字节的内部数据存储器，R0 和 R1 就是其中的两个单元。

需要强调的是，在一般计算机中并没有程序存储器和数据存储器的说法，这是单片机和一般计算机在结构上的区别之一。单片机在空间上将程序和数据分开，单片机所执行的程序代码和单片机执行程序时产生的一些中间结果或临时数据分别存放在程序存储器和数据存储器中，这种结构称为哈佛（Harvard）结构。而在一般计算机中程序和数据共用一个存储空间（如 8086 中只是设立了相应的段，但都是在内存中）。

MCS－51 单片机之所以采用 Harvard 结构的存储器，是由单片机的应用特点决定的。因为单片机往往是为某个特定对象服务，相应的程序设计、调试成功后，只要受控对象功能不变，程序一般保持不变。因此，程序（包括程序中的一些常数）可以而且也应该一次性地、永久地存放到单片机内部，这样不仅可以省去每次开机时的程序重新装入步骤，还可以有效地防止因为掉电或其他干扰而引起的程序丢失和损坏。而程序运行时产生的临时数据在程序结束后或掉电时并不一定要保存，使用 RAM 即可。

3. 外部程序存储器和外部数据存储器

虽然 MCS－51 单片机内部有一定容量的程序存储器和数据存储器，但由于容量有限，在程序很长、中间临时数据量很大的情况下，内部存储器可能不够用。所以单片机有对外部存储器的支持、识别能力，用户可根据需要对存储器进行扩展。

综上所述，MCS－51 单片机的存储器空间在物理上（实际存在的，物理位置不同）可分为 4 个区域：内部程序存储器、外部程序存储器、内部数据存储器、外部数据存储器。但如果从逻辑角度来看，可以把它们看成 3 个独立的存储器空间，分别是：

- 片内和片外程序存储器空间，最大 64 KB。
- 片内数据存储器空间，256 字节，00H～FFH；

➢ 片外数据存储器空间，最大 64 KB。

图 2.1 给出了 MCS－51 单片机的存储器结构。

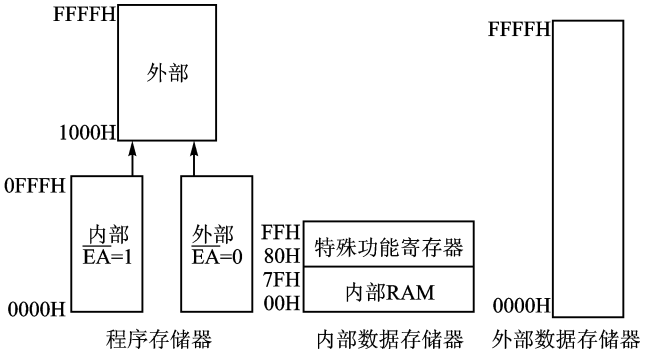


图 2.1　MCS－51 单片机的存储器空间结构

将片内和片外程序存储器看成一个逻辑空间，是因为虽然片内程序存储器在单片机内部，而片外程序存储器“置身”于单片机之外，但如果它们共存则它们的空间地址是连续的。而且用户对它们操作时所使用的指令、操作方法等都基本一样，因此对于用户来说，可以将它们“看成”是一个存储器空间，没有必要分内外。

片内、片外数据存储器则不同。不仅在物理上一个在内部，一个在外部，而且所使用的指令、操作方法等也都不一样（如汇编语言的指令系统中的数据传送指令，访问内部数据存储器要用 MOV 指令，访问外部数据存储器要用专门的 MOVX 指令）。所以片内、片外的数据存储器各占一个逻辑空间，互不干扰，相互独立。

2.1.2　程序存储器

再次强调：程序存储器用于存放程序的机器码（不是源程序），并不存放程序运行时的中间临时数据。它具有只读属性，但也可以向其中写东西，但这种“写”要有特殊的条件，一般用“编程器”来完成这项工作。一旦把单片机装到它的工作位置，进入工作状态，程序存储器中的内容就不能改写了。

关于程序存储器，还要把握以下几点。

1. 内部程序存储器往往是区分一个系列的各种型号单片机的标志

内部程序存储器的不同往往对应了不同的单片机芯片。表 1.1 中有 8031、8051、8751、AT89C51 等型号的单片机，其区别就在于内部程序存储器。

一般来说，单片机内部程序存储器的配置形式有下面几种：

ROM——51 系列中的 8051 芯片中就配置有 4 KB ROM。ROM 就如同练习本，没写程序之前是空白的，可以向其中写东西。可一旦写进去就擦不掉了，所以它只能写一次。要是写错了，就不能再用了。所以只有当产品已经设计定型，确定程序已经调试成功、不需要再作修改的时候，才会使用这种 ROM 配置的单片机。

EPROM——51 系列单片机的 8751 芯片中就有 4 KB EPROM。EPROM 是可用紫外线

擦除的只读存储器。如其中的内容要修改,可以用紫外线擦掉后重新写,这样给用户带来极大的方便。当然其可擦除重写的次数很有限。由于价格高昂(大约为 8051 的 10～15 倍),不太适合大批量使用,比较适合研制产品样机。

内部无 ROM 或 EPROM——既然用 ROM 和 EPROM 都有不便之处,有的单片机就干脆不要内部程序存储器,如 51 系列的 8031 芯片。但程序必须有地方存放,因此该情况下必须外接 EPROM 等芯片来存放程序。虽说要多一两个芯片,但其容量可以视程序长短灵活配置。这比选用 8051、8751 后发现程序存储器不够用的情况要好得多。正因为如此,8031 是十几年前比较流行的一种单片机。

Flash ROM——如现在流行的 AT89C51、AT89S51。Flash ROM 称为闪速存储器,与 EPROM 类似,可以擦掉重写,而且方便得多。它不需要用紫外线,只要用电学方法就可以擦除,而且可擦除次数很多(上千次)。

2. 外部程序存储器的扩展能力

在 MCS－51 单片机中,程序计数器 PC(在微机原理中经常提到)是 16 位的,决定了 CPU 可寻址的程序存储器最大容量是 64 KB(2^{16} KB)。由于在逻辑上可以将内部和外部程序存储器看成一个空间,即内部和外部程序存储器的容量加在一起不能超过 64 KB。

在实际应用中,如果内部程序存储器够用而且比较方便的话(如 AT89C51),没有人会“多此一举”地去扩展程序存储器。即使外扩,一般也不会扩展到 64 KB。

2.1.3 数据存储器

数据存储器在物理和逻辑上均分为两个地址空间:内部和外部数据存储器。片内数据存储器的容量为 256 字节,片外数据存储器的容量最大可扩展到 64 KB。这一点上,51 系列的多数单片机都是相同的。

内部数据存储器是单片机十分重要的资源,实际应用时应首先充分利用内部数据存储器。在大多数工业控制场合,由于不需要进行很复杂的运算,不会产生很多的临时数据,对数据存储器的需求并不大,内部数据存储器通常够用。只有在少数实时数据采集和处理系统中(临时数据量较大),内部 RAM 可能不够,才考虑扩充外部数据存储器。因此对于外部数据存储器,这里不作介绍。

2.1.4 内部数据存储器

MCS－51 单片机的内部数据存储器总容量有 256 字节,地址为 00H～FFH。这 256 字节的地址空间可以分成功能不同的低 128 单元和高 128 单元。

在 256 字节的总容量中,实际上可用的只是全部的低 128 单元和高 128 单元中很少的一部分。因此有的资料上说到“MCS－51 单片机的内部数据存储器是 128 字节”(它是指内部数据存储器的低 128 单元,并不包括高 128 单元)。

1. 内部数据存储器的低 128 单元(00H～7FH)

因为高 128 单元中只用到很少一部分(而且还都有特定用处),所以可以认为:低 128 单

元是真正意义上的内部数据存储器，所以有时干脆称它为“内部 RAM”。

按照用途，又可以将其划分为工作寄存器区、位寻址区、用户 RAM 区。

(1) 工作寄存器区

在这一点上，MCS－51 单片机也和微机原理中介绍的 8086 不一样，这也是单片机的结构特点之一。

MCS－51 单片机的 CPU 中并没有如 8086 CPU 中的 AX、BX 这样的寄存器。但这并不意味着寄存器在单片机中无用武之地。利用寄存器不仅有利于提高单片机的执行速度，而且能提高程序编制的灵活性，简化程序设计。只不过 MCS－51 单片机中的寄存器并不在 CPU 中，而是以内部数据存储器中某个单元的形式出现。

在 MCS－51 单片机的内部数据存储器的低 128 单元中的 00H～1FH 区域，共 32 个单元，对应 32 个寄存器。这 32 个寄存器可以用来暂存运算的中间结果。它们的功能没有明确规定，其作用在某些方面相当于 8086 中的通用寄存器，一般称为工作寄存器，在具体的程序中编程者可根据需要决定它们的作用。

很显然，寄存器是需要名字的，但用 32 个名字命名肯定不方便。于是 51 单片机中将它们划分为 4 个组，每组包含 8 个寄存器，都称为 R0～R7。这样，每组 R0～R7 分别占有 00H～07H、08H～0FH、10H～17H、18H～1FH 对应的单元。

在 1.5 节任务 1 中用到的“R0、R1”就是第一组中的 2 个工作寄存器，分别占用地址为 00H 、01H 单元。

注意： 对于这 32 个单元中的某一个单元，可以用 Rn(n＝0～7)的形式去访问它，也可以用其单元地址的形式去访问它。如“MOV R0，A”和“MOV 00H，A”。两条指令的功能是相同的，但两者对该单元的寻址方式不同，指令的机器码不同，故执行速度也不同。

(2) 位空间

在内部数据存储器中，还有一个“位空间”。这种位处理能力是 MCS－51 单片机的一个重要特点，在开关量的控制中非常具有优势。前面提到过，单片机内部有针对控制而设计的结构，位处理器就是其中之一。有了位处理器，就可以将控制场合中很多由“门电路”硬件完成的逻辑操作功能，直接用“位操作指令”进行处理。方法简便，同时也免去了没有位处理器时过多的数据往返传送、字节屏蔽和测试分支，大大地简化了编程，也增强了控制的实时性。

位空间是可以直接进行位操作的所有“位”的集合，这些“位”分布在内部数据存储器中。这个“位空间”不是独立的，其中一部分“位”处于低 128 的部分单元中，另一部分“位”处于高 128 的部分单元中。也就是说，对于这些单元，既可以对其按字节操作，也可以对某单元中的某一位单独进行“位”操作。

其中，在低 128 字节中的 20H～2FH 这 16 个单元中共计 128(16×8)个连续的位，它们的位地址也是连续的(00H～7FH)。当然，20H～2FH 这 16 个单元也可以作为一般 RAM 单元使用，进行字节操作。

表 2.1 所列为这 128 个位的分布和及其位地址。

表 2.1　内部 RAM 中的 128 个位

单元地址	高位 MSB			位地址			低位 LSB	
2FH	7FH	7EH	7DH	7CH	7BH	7AH	79H	78H
2EH	77H	76H	75H	74H	73H	72H	71H	70H
2DH	6FH	6EH	6DH	6CH	6BH	6AH	69H	68H
2CH	67H	66H	65H	64H	63H	62H	61H	60H
2BH	5FH	5EH	5DH	5CH	5BH	5AH	59H	58H
2AH	57H	56H	55H	54H	53H	52H	51H	50H
29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
28H	47H	46H	45H	44H	43H	42H	41H	40H
27H	3FH	3EH	3DH	3CH	3BH	3AH	39H	38H
26H	37H	36H	35H	34H	33H	32H	31H	30H
25H	2FH	2EH	2DH	2CH	2BH	2AH	29H	28H
24H	27H	26H	25H	24H	23H	22H	21H	20H
23H	1FH	1EH	1DH	1CH	1BH	1AH	19H	18H
22H	17H	16H	15H	14H	13H	12H	11H	10H
21H	0FH	0EH	0DH	0CH	0BH	0AH	09H	08H
20H	07H	06H	05H	04H	03H	02H	01H	00H

(3) 用户 RAM 区

在低 128 单元的内部 RAM 中，工作寄存器占用了 32 个单元，位寻址区占用 16 个单元，剩下 80 个单元没作任何特殊规定，就是供编程者自由使用的 RAM 区，其地址为 30H～7FH。

应用中通常把数据缓冲区和堆栈开辟在此区中。虽然原则上堆栈可以是片内 RAM 的任意区域，但具体应用时堆栈的设置要和 RAM 的分配统一考虑。在具体的程序中，00H～1FH 单元可能已被用作工作寄存器，20H～2FH 中的某些单元也可能用作位变量了，因此堆栈区域一般设在 30H 以后为宜。当然如 20H～2FH 中只用了 20H 单元中的几个位，其他单元并没有被用作位变量区，堆栈也完全可以从 21H 单元开始。

内部数据存储器的低 128 单元分配如图 2.2 所示。

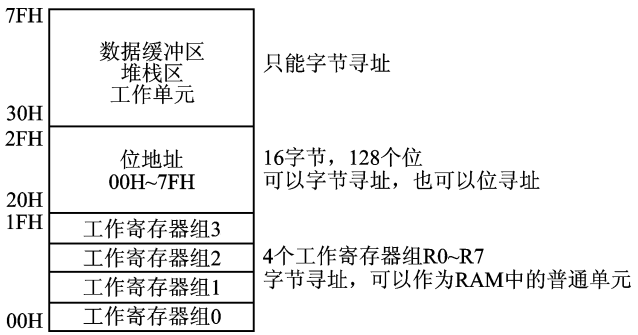


图 2.2　内部数据 RAM 的分配

2. 内部数据存储器的高 128 单元(80H～FFH)

如前所述,在 MCS－51 单片机内部除 CPU、并行 I/O 口、ROM、RAM 外,还有定时/计数器、串行 I/O 口、中断系统等功能部件。为了控制这些部件的工作状态和工作过程,在 MCS－51单片机中又专门设置了一些特殊功能寄存器,称为 SFR(Special Function Register)。相对于功能通用的工作寄存器,这些寄存器的功能有专门的规定,使用时必须严格遵循它的功能和格式定义。

MCS－51 单片机内的 SFR 能综合、实时地反映整个单片机内部各个部件的工作状态和工作方式。利用 SFR 可以很方便地完成对定时器、串行口、中断逻辑的控制。因此掌握各个 SFR 的工作方法,对于实现对单片机应用系统的控制具有很重要的意义。

(1) 高 128 单元中的 21 个 SFR

内部数据存储器的高 128 单元就是专门供给 SFR 使用的,所以该区域称为 SFR 区。MCS－51 单片机的 51 子系列定义了 21 个 8 位的 SFR。这 21 个 SFR 是离散分布的,地址并不连续。目前只定义了 21 个,不可能占满全部的高 128 字节。对高 128 单元中未定义为 SFR 的单元,编程者是不能使用的。当然,随着 51 系列单片机的发展,在高 128 单元中可能还会另外定义出相关的 SFR。如 52 子系列比 51 子系列多出一个定时器 2,于是就多定义了 5 个与定时器 2 有关的 SFR(T2CON、TH2、TL2、RLDH、RLDL)。

这 21 个 SFR 与各控制功能部件的关系(有所交叉)如下:

- CPU:A、B、PSW、DPH、DPL(组成 DPTR)、SP;
- 定时/计数器:TH0、TL0(组成 T0)、TH1、TL1(组成 T1)、TMOD、TCON;
- 中断系统:IE、IP;
- 并行口:P0、P1、P2、P3;
- 串行口:SCON、PCON、SBUF。

注意: 单片机内部还有一个寄存器,也就是程序计数器 PC。虽然它的功能也很特殊(存放下一条将要被执行的指令的地址),但它不同于这里所讲的寄存器。PC 没有地址,在物理上是独立的。因此,PC 没有被列在 SFR 中。

表 2.2 给出了 21 个 SFR 的名称和地址分布。

表 2.2　SFR 的分布地址和位空间

SFR	高位 MSB		位地址/位符号			低位 LSB			字节地址
B	F7H	F6H	F5H	F4H	F3H	F2H	F1H	F0H	F0H
ACC	E7H	E6H	E5H	E4H	E3H	E2H	E1H	E0H	E0H
PSW	D7H	D6H	D5H	D4H	D3H	D2H	D1H	D0H	D0H
	CY	AC	F0	RS1	RS0	OV	F1	P	
IP	BFH	BEH	BDH	BCH	BBH	BAH	B9H	B8H	B8H
	—	—	—	PS	PT1	PX1	PT0	PX0	
P3	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H	B0H
	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	

续表 2.2

SFR	高位 MSB			位地址/位符号			低位 LSB		字节地址
IE	AFH	AEH	ADH	ACH	ABH	AAH	A9H	A8H	A8H
	EA	—	—	ES	ET1	EX1	ET0	EX0	
P2	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H	A0H
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
SBUF									(99H)
SCON	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H	98H
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
P1	97H	96H	95H	94H	93H	92H	91H	90H	90H
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	
TH1									(8DH)
TH0									(8CH)
TL1									(8BH)
TL0									(8AH)
TMOD	GATE	C/－T	M1	M0	GATE	C/－T	M1	M0	(89H)
TCON	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H	88H
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
PCON	SMOD	—	—	—	GF1	GF0	PD	IDL	(87H)
DPH									(83H)
DPL									(82H)
SP									(81H)
P0	87H	86H	85H	84H	83H	82H	81H	80H	80H
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	

从表 2.2 中可以看出，每一个 SFR 都有一个固定的字节地址(最右一列)，还有一个符号名(最左一列)，用户都可以使用(当然常用的是符号名，这样更直观)。

(2) 特殊功能寄存器中的位地址空间

在 21 个 SFR 中，也有一部分可以进行位处理(相对于前面提到的 128 个位，这些位都有特殊定义，以下就称为“特殊功能位”)。表 2.2 同时给出了其中能进行位寻址的 SFR 的名称和位地址。

51 子系列中有 11 个的 SFR(表中字节地址没有带括号的)中的某些位可以位寻址，它们是：B、ACC(A)、PSW、IP、P3、IE、P2、SCON、P1、TCON、P0。其中 IP 和 IE 的 5 位不能位操作(功能未定义)，所以在 SFR 中的特殊功能位有 $11\times8-5=83$ 位，它们的位地址也不连续。加上前面提到的内部 RAM 中的 128 个位，内部数据存储器可以提供包含 211 个位的位空间。

除了 A、B 中的 16 个位以外，其他的特殊功能位都有一个位符号名。

注意：位地址空间中一部分位于低 128 单元，一部分位于高 128 单元，也就是位空间与内部 RAM 和 SFR 重叠。而且每一个可以位处理的位也有一个位地址，在形式上可能与单元地

址相同(如有 08H 单元,也有 08H 位),这时就要依靠指令的性质来判断这个地址是单元的地址还是位的地址。

如果是字节操作指令,如“MOV A,08H”指令,08H 表示的是单元地址;

如果是位操作指令,如“MOV C,08H”指令,08H 表示的是位地址。根据表 2.1 可知,这是 21H 单元的最低位的地址。

2.1.5 几个特殊功能寄存器简介

在 21 个 SFR 中,与定时/计数器、串行口和中断系统相关的 SFR 将在后续相关章节中进行详细介绍,这里先介绍与 CPU 的运算和控制直接相关的几个 SFR。

1. 累加器 ACC

累加器 ACC 又称 A 寄存器,其功能不能从名字“累加器”上理解,它并不是一个做加法的部件,而只是一个 8 位寄存器。它是汇编语言程序编写中使用最频繁的寄存器之一。大家在附录 A 的汇编语言指令表中可以看到,很多的指令都离不开它:大多数单操作数指令的操作数取自于 A,许多双操作数指令的一个操作数取自于 A,算术运算的运算结果也基本上存放在 A 中。

累加器在指令中有 A 和 ACC 两种写法,用在不同的指令和场合(详见附录 A)。

2. B 寄存器

B 寄存器主要用于乘法和除法运算中,与累加器 ACC 配合使用。对于其他指令,B 寄存器可视为内部数据存储器中的一个普通单元,可以任意使用。

3. 程序状态字寄存器 PSW

程序状态字寄存器 PSW 用于存放 CPU 执行程序时的某些程序状态信息,程序员可以从中了解到 CPU 的当前状态,并作出相应的处理。PSW 中有些位的状态是根据程序的执行结果由硬件自动设置的,用户不能直接修改。而有些位的状态则可以使用软件来设定。

PSW 中各特殊功能位的含义如下:

CY	AC	F0	RS1	RS0	OV	—	P

(1) CY

进(借)位标志,以下均简化为 C 来表示。MCS-51 单片机中的运算器是一种 8 位的运算器,C 表示在做运算时运算结果最高位 D7 向高位字节是否有进(借)位,也就是表示运算结果是否超过了 8 位无符号数的表示范围 0～255。如果超出,则 C 为 1。

例如:87H(135)+9AH(154)=121H(289)(1000,0111+1001,1010=1,0010,0001),运算结果放在 A 中的是 21H,最前面的 1 就成了进位,则 C=1。

在很多场合,用 C 的状态来进行两个数据大小的比较。

(2) AC

辅助进位标志。表示在进行加(减)法运算时,运算结果低 4 位向高 4 位是否进(借)位。如果有,则 AC 位为 1。

例如：57H +3AH=91H(0101,0111+0011,1010)，AC=1。

AC 主要用于二-十进制调整。

(3) F0

用户自定义的标志，无具体定义。编程人员可以根据具体程序中的需要来定义它的含义和使用方法。

(4) RS1、RS0

在内部 RAM 的 4 组工作寄存器中，在任何时刻 CPU 只能访问其中一组，其他三组不能同时访问。当前能够访问的那一组工作寄存器称为“当前寄存器组”。

程序中可以通过 RS1、RS0 两位来进行当前工作寄存器组的选择。对应于 00～11 的组合，分别用于选择工作寄存器区 0、1、2、3。其对应关系如表 2.3 所列。

表 2.3　当前工作寄存器组选择

RS1　RS0	当前寄存器组	对应单元地址
0　0	工作寄存器组 0	00H～07H
0　1	工作寄存器组 1	08H～0FH
1　0	工作寄存器组 2	10H～17H
1　1	工作寄存器组 3	18H～1FH

也就是说，Rn 可能代表不同的单元。比如当 RS1 和 RS0 为 00 时，R0 就代表 00H 单元；而当 RS1 和 RS0 为 01、10、11 时，R0 又分别代表 08H、10H、18H 单元。

(5) OV

溢出标志。在对带符号数做加减运算时(注意是带符号数据)，用来表示运算结果是否超出了 8 位带符号数的范围(－128～＋127)。如超出，则 OV 为 1。

在乘法运算中，OV 为 1 表示乘积超出了 255；

在除法运算中，OV 为 1 表示除数为 0。

(6) P

奇偶标志，表示累加器 A 的 8 位中为“1”的位数的奇偶性。A 值中为 1 的位数为奇数，则该位为 1。执行任何一条与 A 相关的指令，A 中 1 的位数都有可能发生变化，P 也随之变化。如：

A＝F2H，P＝1
A＝E2H，P＝0　} 两者 A 都是偶数，但 P 不同

A＝F1H，P＝1
A＝E1H，P＝0　} 两者 A 都是奇数，P 也不同

奇偶标志常用在串行通信中，如用奇偶校验的方法来检验数据传输的可靠性。

在 7 个标志位中，AC、P、OV 标志由硬件置位或清除，用户不能直接修改。标志位 C、F0、RS1、RS0 可以通过位操作指令进行设置，如“CLR C”可以将 C 清 0。

4. 数据指针 DPTR

DPTR 由两个 8 位寄存器 DPH、DPL 组成。可以将它当作 16 位寄存器来处理，也可以将它分成两个 8 位寄存器来处理。

之所以称之为“数据指针”，是因为使用时都是用它指向存储器中所需要的某个数据。如当访问外部 RAM 中的某个单元时，其地址是 16 位的，就需要将该地址存放在 DPTR 中以指向该单元。当访问程序存储器时，DPTR 用作基地址寄存器，和 A 一起指向存放在程序存储器中的某个字节。

5. 堆栈指针 SP

日常生活中，我们都会注意到这样的现象：家里洗好的碗一只一只摞起来，最迟放上去的放在最上面，而最早放上去的则放在最下面。在取的时候正好相反，先从最上面取。其实这种现象比比皆是，如枪膛里的子弹、建筑工地上堆放的砖头、材料，仓库里放的货物等。这些现象实际上是一种存取物品的规则，可以用一句话来概括：“先进后出”或“后进先出”。

在计算机中，也可以在 RAM 中构造这样一个区域，用来存放数据。这个区域称之为“堆栈”，存放数据的规则就是“先进后出”或“后进先出”。

51 系列单片机中的堆栈是设置在内部 RAM 中的。由于 MCS－51 单片机中内部数据存储器的容量很有限，每个程序对堆栈的大小需求也不同。因此不能固定地拿出一块空间用作堆栈，于是就把分配堆栈的权利给编程者，根据需要自行确定。

也就是说，51 单片机中堆栈的位置是可以变化的，这种变化就体现在程序中对 SP 内容的设置。只要在程序某处更改了 SP 的值，就可以把堆栈设置在规定的内存单元区域中。如在程序开始时使用“MOV SP，＃5FH”指令，就是把堆栈区域设置在从内部 RAM 中从 60H 开始的区域中。

除用 MOV 指令改变 SP 的内容外，在执行 PUSH、POP、中断响应与返回、子程序调用与返回等指令时，SP 的值都会自动增减。但无论如何变化，SP 的内容始终是堆栈顶部的地址，即堆栈顶部在内部 RAM 中的位置。

堆栈是内部 RAM 的一部分，但并不意味着该区域成为一种专用内存，它仍然可以像普通内存区域一样使用。但既然把它设置成堆栈区，编程者自然就不会把它当成普通内存来用。这个区域的特殊之处，在于存放和取用数据的方式（后进先出），另外有一个专用的指针 SP 指向要处理的数据。

2.2 MCS－51 单片机的引脚信号

在完成 1.5 节任务 1 的过程中，我们搭建了一个单片机应用系统的硬件电路，连了一些线。因为是刚开始，所以只是“照葫芦画瓢”，至于为什么要那么连线并没有作出说明。

在实际的应用系统开发过程中，在设计和制作硬件电路时就是将被控对象通过一定的电路连接到单片机某个引脚上。这时就必须要知道单片机每一个引脚的具体功能。当然每个引脚在所有引脚中是第几个、什么位置都是固定的，没有必要去死记硬背，只要在用的时候查一下相关资料（如图 1.5 所示）即可。

在图 1.5 中，有些引脚的功能有两个。由于工艺及标准化等原因，MCS－51 系列把芯片引脚数目定为 40 个。但 51 单片机为实现其所有功能所需要的信号数目却远远超过这个数。如何才能解决这个“供需”矛盾呢？唯一可行的办法，就是给其中的部分引脚赋以双重功能（即基本功能和第二功能）。

2.2.1 MCS－51 单片机引脚的基本功能

1. 电源线

MCS－51 单片机使用单一的＋5 V 电源(当然电压在小范围内变化也是可以的)。从图 1.5 中可以看到,第 40 脚 Vcc 是电源接入引脚,一般接＋5 V 电源。第 20 脚 Vss 是接地引脚。这两个引脚在 1.5 节任务 1 中已经接触过。

2. 外接晶体振荡器的引脚

单片机是一种时序电路,必须提供稳定的脉冲信号才能使其正常工作。由于不同的应用场合对单片机的速度要求不一样,因此单片机内部并没有集成晶体振荡器,而是由用户根据自己的情况去选择外接。外接的晶体振荡器的振荡信号必须经过放大、处理才能形成能使单片机工作的时钟信号,因此在 MCS－51 单片机的内部集成有这部分电路。XTAL1(第 19 脚)和 XTAL2 引脚(第 18 脚)就是这部分电路的输入端和输出端。

这两个引脚在 1.5 节任务 1 中也接触过,我们在这两个引脚之间接了晶体振荡器。关于具体怎么接,在 2.3 节的振荡电路中有详细介绍。

3. P0～P3 口的引脚

在 1.5 节任务 1 中,用到了 P1.0～P1.7 这 8 个脚。在图 1.5 中还可以看到,除 P1.0～P1.7 外,还有 P0.0～P0.7、P2.0～P2.7、P3.0～P3.7。这 32 个脚都可以作为输入/输出使用。当然它们的功能远不止“使 8 个 LED 闪烁”。

这 32 个脚分别组成了与单片机外部联系的 4 个并行 I/O 口。它们的基本功能都是作为通用的 I/O 口(如连接一个按键、连接一个 LED),完成高低电平的输入、输出。但更常用的功能是它们的第二功能(见 2.2.2 小节)。

4. 控制信号引脚

① ALE(第 30 脚):地址锁存允许输出信号。

当 MCS－51 单片机外部扩展了存储器或一些接口芯片后,在访问外部存储器某个单元或访问接口芯片中某个端口时,都需要输出 16 位地址信号用来选中该存储单元或端口。在 MCS－51 单片机中规定用 P0、P2 口的 16 根线输出这个 16 位地址信号。其中 P0 口输出地址的低 8 位,P2 口输出地址的高 8 位(这是 P0、P2 口的第二功能)。

单片机根据该 16 位地址找到相关单元后,对这个单元进行读/写的数据也要从 P0 口输入、输出。这样,P0 口一方面要输出低 8 位地址,还要输入/输出要读/写的数据。P0 口上的信息何时是地址、何时是数据,就需要根据 ALE 引脚上电平的高低加以区别。

规定 ALE 引脚为高电平期间,P0 口上的信息就被“看作”地址信息;ALE 引脚为低电平期间,P0 口上的信息就被“看作”数据信息。但由于 P0 口在输出地址低 8 位和输入/输出数据这 2 个动作不可能同时进行,在输入/输出数据时地址信息肯定已经不存在,所以要另外提供一个器件用于保存地址信息,这个器件就是“地址锁存器”。利用在 ALE 引脚输出脉冲的下降沿,将 P0 口上的信息(此前 P0 口上的信息是地址信息)锁存到地址锁存器中,这样就实现了低 8 位地址和数据的隔离。

而 P2 口并不是数据、地址复用的，所以不需要这样处理。

ALE 引脚信号频率固定为晶振频率的 1/6。即使不访问外部存储器或外部端口，ALE 引脚信号也会同样出现。因此某些场合可以将 ALE 引脚信号作为外部时钟或外部定时脉冲使用（如经处理后可作为 ADC0809 的 CLK 信号）。

② RST（第 9 脚）：复位信号，用于单片机的初始化操作。

用于连接外部电路产生“复位”信号。这个引脚在 1.5 节任务 1 中已接触过。其复位原理及相关知识参见 2.3 节的介绍。

③ $\overline{\text{PSEN}}$（第 29 脚）：外部程序存储器读信号（存储器输出允许信号）。

从片外程序存储器读取指令或常数时，用于命令外部程序存储器做输出动作。$\overline{\text{PSEN}}$信号和 ALE 引脚信号同频率，两者相互配合。

注意：与 ALE 引脚信号不同，在访问内部程序存储器时，并不产生$\overline{\text{PSEN}}$信号。因此在不扩展外部程序存储器时，$\overline{\text{PSEN}}$很少用到。

④ $\overline{\text{EA}}$（第 31 脚）：内部和外部程序存储器的访问控制信号。

虽然从逻辑上将内部和外部程序存储器看作一个空间，但毕竟一个在内，一个在外。何时执行内部程序存储器中的指令、何时执行外部程序存储器中的指令，需要$\overline{\text{EA}}$信号来控制。

当$\overline{\text{EA}}$为高电平时，既可访问内部程序存储器，也可访问外部程序存储器。若 PC 值在内部程序存储器容量范围内，就执行内部程序存储器中的程序。反之，自动转去执行外部程序存储器的程序。

当$\overline{\text{EA}}$为低电平时，只访问外部程序存储器，而不管是否存在内部程序存储器。如果使用没有内部程序存储器的 8031，$\overline{\text{EA}}$就必须接地，强制单片机访问外部程序存储器。

在实际应用时，一般不会让内部和外部程序存储器“共存”（不方便）。如果内部程序存储器够用而且方便（如 AT89C51），就不要扩展外部程序存储器，此时$\overline{\text{EA}}$接高电平。如内部程序存储器不够用，就干脆选用内部不带程序存储器的单片机（如 8031），外部程序存储器全部外接，此时$\overline{\text{EA}}$接地。

在 1.5 节任务 1 中，由于使用的是 AT89C51 芯片中的 4 KB 程序存储器，因此将$\overline{\text{EA}}$引脚接高电平。

2.2.2 MCS－51 单片机引脚信号的第二功能

对于同一系列中相同封装形式的单片机，其引脚的基本功能相同，不同的是引脚的第二功能。引脚的第二功能有时比第一功能还有用。若缺少第二功能，甚至可能导致系统不能工作。

1. I/O 口的第二功能

P0～P3 口的基本功能都是作为通用双向 I/O 口，用来输入或输出数据。它们的第二功能分别是：

- P0 口（AD0～AD7）：第二功能是在访问外部存储器或外部接口芯片时，为分时使用的低 8 位地址输出和 8 位数据总线的输入/输出。
- P1 口：在 MCS－51 单片机的应用系统中，P1 口一般都作为通用 I/O 口使用。
- P2 口（A8～A15）：第二功能是在访问外部存储器或外部接口芯片时，输出高 8 位

地址。

➢ P3 口：P3 口的 8 个引脚都具有不同的第二功能，而且都是很重要的功能。

初学者往往对 P0、P2 和 P3 口第二功能用法的迷惑不解，总认为需要用某条指令来完成第一功能和第二功能的切换。事实并非如此。如 P3.6 和 P3.7，当单片机外接 RAM 或有外部 I/O 口时，硬件电路上自然要将这两个引脚连接到该外部 RAM 芯片的写引脚和读引脚上。在执行相关读/写指令时，P3.6 和 P3.7 就会自动被用作第二功能（$\overline{WR}$和$\overline{RD}$），自动充当着传输“写”或“读”信号的作用。此时自然不可能再用作通用 I/O 口了。这些并不需要事先用指令说明，在硬件设计中就已经处理好了。

表 2.4 给出了 P3 口的第二功能。

表 2.4　P3 口的第二功能

引　脚	信号名称	第 二 功 能
P3.0	RXD	串行接收时，串行数据的输入口
P3.1	TXD	串行发送时，串行数据的输出口
P3.2	$\overline{INT0}$	外部中断 0 的中断请求信号输入端
P3.3	$\overline{INT1}$	外部中断 1 的中断请求信号输入端
P3.4	T0	定时器 0 作为外部计数时，外部计数脉冲输入端
P3.5	T1	定时器 1 作为外部计数时，外部计数脉冲输入端
P3.6	$\overline{WR}$	访问外部数据存储器写信号输出端
P3.7	$\overline{RD}$	访问外部数据存储器读信号输出端

P3 口的第二功能信号都是单片机的重要控制信号。在实际使用时都是根据系统需要选用其第二功能，其余的可以用作第一功能进行数据的 I/O。

2. EPROM 程序存储器固化所需要的信号

① Vpp（与$\overline{EA}$引脚复用）：EPROM 型单片机进行内部 EPROM 编程时，+5 V 的电压肯定不够（否则正常工作时也能修改它），应该提供电压更高的电源。此引脚可连接 +25 V 的编程电源电压。

② $\overline{PROG}$（与 ALE 引脚复用）：程序代码写入程序存储器时需要指定一个速度，一般是一个编程脉冲写入 1 个字节。编程脉冲就来自于$\overline{PROG}$引脚。

用户设计的应用系统电路板上用不到这两个信号。使用编程器 写入程序代码时，编程器已经将这两个信号处理好，因此用户也不必关心。

3. 备用电源输入 Vpd

在计算机运行过程中，可能会遇到掉电而导致一些不良后果。单片机也是如此，如果没有后备电源，单片机在掉电期间就会丢失一些重要的中间数据（当然程序代码不会丢失）。

备用电源引脚与 RST 脚复用。如果该脚接有备用电源，一旦发生断电、电位突降低于规定的电平，而 Vpd 又在规定的电压范围内，就会自动通过 Vpd 向单片机供电，以保证程序正常运行。

2.2.3 AT89C2051 单片机简介

除标准 40 引脚的 AT89C51 外，C51 系列还有一种精简型的 AT89C2051 芯片。图 2.3 所示为 AT89C2051 引脚图。

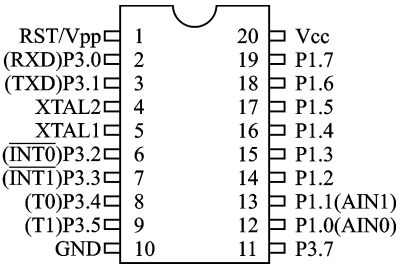


图 2.3 AT89C2051 引脚图

AT89C2051 和 AT89C51 的指令系统兼容引脚功能和最小系统的构成也一样。与 AT89C51 相比，AT89C2051 具有以下几个特点：

① 20 引脚封装，体积更小，成本更低。

② 片内具有 2 KB 的 Flash ROM。

③ 片内带模拟比较器，P1.0 和 P1.1 除用作通用 I/O 以外，还可作为模拟比较器的正输入端和负输入端，与片内模拟比较器相连。P3.6 用作模拟比较结果输出，没有外部引脚。但程序中可通过查询 P3.6 可获知 P1.0 和 P1.1 的比较结果；

④ 只有两个并行接口(P1、P3)，并行口线变得很少，可利用资源比较紧张。由于没有 P0 和 P2 口，因此不能外接外部存储器，也不能外扩接口，不允许使用 MOVX 指令。所以 AT89C51 能运行的程序不一定能直接移植到 AT89C2051 上。

由此可见，AT89C2051 特别适合小系统，为很多非常简单的嵌入式控制应用提供灵活且价格适宜的方案。

2.3 MCS－51 单片机的振荡电路和复位电路

在第 1 章中曾说到：待程序仿真运行无误且将程序机器代码烧写到程序存储器中后，将单片机芯片插到用户电路板上，接通电源，一般情况下系统就可以正常地脱机运行了。但因为使用仿真器仿真时，使用的是仿真器本身的振荡电路和复位电路，并没有使用应用系统电路板上的振荡电路和复位电路，因此如果应用系统的振荡电路和复位电路有问题(不起振或不能产生复位电平)，系统必然也不能运行。

2.3.1 振荡电路

单片机的工作步调很像一个大钟，什么时候时针动、什么时候分针动、什么时候秒针动、都有严格的规定，一点也不能乱。

单片机在执行指令时，通常将一条指令分解为若干基本的微操作，这些微操作对应的脉冲信号在时间上的先后次序，称为时序。为了让单片机内部各部件按一定的步调工作，则必须在唯一的时钟信号的控制下，严格地按照时序进行工作。

MCS－51 单片机内部有一个时钟电路(其核心是一个反相放大器)，但并没有形成时钟的振荡信号，因此必须外接谐振器才能形成振荡。如何使用这个内部放大器，可以根据不同的场合作出不同的选择。这样就对应了单片机时钟产生的不同方式：若采用这个放大器，即为内

部方式；若采用外部振荡输入，即为外部方式。

1. 内部方式

如果在 MCS－51 单片机的 XTAL1 和 XTAL2 引脚之间外接晶体谐振器，便会产生自激振荡，可在内部产生与外加晶体同频率的振荡时钟。用示波器可以观察到 XTAL2 输出的时钟信号。

最常见的是采用如图 2.4 所示的振荡电路，这在 1.5 节任务 1 中已经接触过。不论是 HMOS 还是 CHMOS 工艺的单片机，这个电路都是一样的。

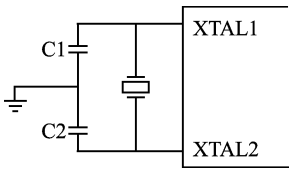


图 2.4　单片机的振荡电路

不同单片机的最高工作频率不一样，如 AT89C51 的最高工作频率为 24 MHz，AT89S51 的最高工作频率可达 33 MHz。由于制造工艺的改进，现在单片机的工作频率范围正向两端延伸，可达 40 MHz 以上。振荡频率越高表示单片机运行的速度越快，但同时对存储器的速度和印刷电路板的要求也就越高。有时频率太高反而会导致程序不好编写（如延时程序）。一般来说，不建议使用很高频率的晶体振荡器。MCS－51 单片机应用系统一般都选用 6～12 MHz 的晶体振荡器。

这个振荡电路对 C1、C2 的值没有严格要求，但电容值的大小多多少少会影响晶体振荡器的稳定性、振荡器频率的高低、起振的快速性等。一般外接晶体时，C1、C2 的值通常选为 20～100 pF。

另外在设计电路板时，晶体振荡器、电容等均应尽可能靠近单片机芯片，以减小分布电容，进一步保证振荡器的稳定性。

2. 外部方式

在较大规模的应用系统中可能会用到多个单片机，为保证各单片机之间时钟信号的同步，应当引入唯一的公用外部脉冲信号作为各单片机的共同的振荡脉冲，也就是采用外部方式，外部振荡信号直接引入 XTAL1 和 XTAL2。

由于 HMOS 和 CHMOS 单片机内部时钟进入的引脚不同，因此外部振荡信号的接入方式也不一样。外部方式不是应用的重点，这里不再叙述。

2.3.2　时序定时单位

在 1.5 节任务 1 中使用的延时子程序可以延时一段时间。要想知道这样的延时程序能够延时多长时间，就要初步了解时序方面的相关知识。

所谓时序，用通俗的话说就是 CPU 执行某条指令时所产生的相关信号的维持时间以及先后顺序。时序是用定时单位（时间基准）来说明的。MCS－51 的定时单位有 4 个：振荡周期、时钟周期、机器周期、指令周期。

（1）振荡周期 P

就是晶体振荡器产生的振荡信号的周期，通常定义为节拍（P）。

（2）时钟周期 S

外部振荡电路形成的振荡信号在内部时钟电路中经过二分频后，成为单片机的时钟周期，

也称为状态周期(S)。1 个时钟周期包含 2 个节拍,其前半个周期称为节拍 1(P1),后半个周期称为节拍 2(P2)。

(3) 机器周期

计算机访问一次存储器需要的时间,称之为一个机器周期 T_p,也是计算机指令执行的最小时间单位。在 MCS-51 单片机中,规定一个机器周期包含 6 个状态周期(S1～S6),也就是 12 个节拍,分别记作 S1P1、…、S6P1、S1P2、…、S6P2。因此,机器周期就是振荡周期的 12 倍。

如晶振频率为 12 MHz,则振荡周期为 1/12 μs,T_p 就是 1 μs。如果使用 6 MHz 的晶振,T_p 就是 2 μs。

(4) 指令周期

MCS-51 单片机的所有指令中,有的完成得比较快,有的完成得比较慢。为了衡量指令执行时间的长短,又引入了“指令周期”的概念。指令周期就是指执行某一条指令的时间,由若干个 T_p 组成。需要 T_p 数越少的指令执行速度越快,反之执行速度越慢。

MCS-51 单片机每一条指令都有固定的指令周期。执行需要一个 T_p 的指令称为单周期指令,需要两个 T_p 的指令称为双周期指令。MCS-51 单片机的指令通常包括单周期、双周期和四周期指令 3 种。

这些数据,大部分不需要我们去记忆,但是有一些是要知道的,如 DJNZ 指令是双周期指令。1.5 节任务 1 中的延时过程主要就是通过反复执行两条 DJNZ 指令完成的。

2.3.3 延时程序分析

在 1.5 节的任务 1 和 1.6 节的任务 2 中,都使用了这样的程序:

```
        MOV     R0,#0FFH
D1:     MOV     R1,#0FFH
D2:     DJNZ    R1,D2
        DJNZ    R0,D1
```

通过存储器的学习,我们已经知道程序中的符号 R0、R1 是两个工作寄存器,代表了两个 RAM 单元,可以用来存放一些中间数据。那么这个子程序又是如何实现延时的?延时的时间又是多少呢?下面我们来分析程序执行过程。

MOV:这条指令的意思是传递数据。“MOV R0,#0FFH”的功能是将数据 FFH 送到 R0 中去。因此执行完这条指令后,R0 单元中的值就应当是 FFH。在 FFH 前面有个“#”号,是用来说明 FFH 就是一个数据。“MOV R1,#0FFH”的意思也是一样。

DJNZ:这条指令后面跟着的两个参数,一个是 R1,一个是 D2。R1 已经知道是一个工作寄存器,D2 在 1.5 节任务 1 的实现中已学过,称为“标号”。DJNZ 指令执行时是先将其后面的第一个参数中的数值减 1,然后判断它是否等于 0:如果等于 0,就往下执行;如果不等于 0,就转移到第二个参数所指定的地方去。很显然,本条指令的最终执行结果就是,在原地转圈 FFH(255)次。

执行完“DJNZ R1,D2”之后(也就是 R1 的值等于 0 之后),就会去执行下面一行,也就是“DJNZ R0,D1”。大家自行分析一下,就可以明白这条指令执行的结果就是 R0 中的值减 1,如果不为 0 则转去执行“MOV R1,#0FFH”。然后“DJNZ R1,D2”指令又将被执行 255 次。这

样“DJNZ R1,D2”指令总共执行了 255×255=65025(次)。

另外经过分析,指令“MOV R1,#0FFH”要执行 255 次,指令“DJNZ R0,D1”也要执行 255 次。

同一条指令执行这么多次,目的当然就是延时(实际上是耗时)。虽然某条指令执行一次需要的时间很短,但如果让 CPU 反复、多次执行某几条指令,通过“积少成多”就可以实现一定长度的时间延时。

2.3.4 复位电路

复位是使 CPU 和系统中的其他功能部件都处于一个确定的初始状态,复位后计算机就从这个状态开始工作。在复位期间,CPU 并没有开始执行程序,而是在做准备工作。

无论是在计算机刚上电、断电后,还是系统出现故障时,都需要复位。

1. MCS-51 单片机复位条件

MCS-51 单片机的复位靠外部电路实现。当时钟电路工作时,只要在单片机的 RST 引脚(第 9 脚)上持续出现 2 个 T_p 以上的高电平就可以使单片机复位。但时间过短往往使复位不可靠。为了确保复位,RST 引脚上的高电平一般要维持大约 10 ms 以上。

2. 常见的复位电路

实际上,PC 机的两种启动方式(冷启动和热启动)就对应了两种不同的复位方式。一个是在计算机没有工作时,通过给计算机加电实现复位。另一个是在计算机已经开始工作的情况下,通过重新启动实现复位。

同样,MCS-51 单片机的复位按原理也可分成上电复位和按键手动复位两种。图 2.5 是这两种复位电路的示意图。

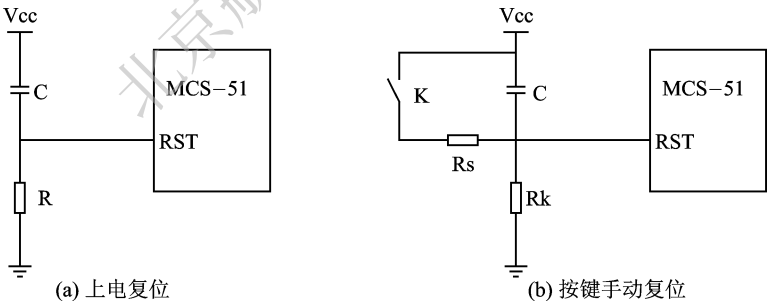


图 2.5 单片机常见的复位电路

(1) 上电复位电路

如图 2.5(a)所示,上电复位是利用电容充电来实现的。在接通电源的瞬间,RST 端的电位与 Vcc 相同,都是+5 V。随着 RC 电路的充电,RST 的电位逐渐下降,只要保证 RST 为高电平的时间大于 10 ms 就能正常复位了。

在 1.5 节任务 1 中,已经连接过这种上电复位电路。当时接了一个 22 μF 的电解电容,接了一个 1 kΩ 的电阻。

(2) 按键复位电路

在单片机已经通电的情况下,只需要按下图 2.5(b)中的 K 键就可以复位。此时电源 Vcc 经过电阻 Rs、Rk 分压,在 RST 端产生一个复位高电平。如 Rk 取 1 kΩ,Rs 取 200 Ω 左右,这样当按下 K 键时 RST 端的电压约为 4.2 V,满足高电平要求。

在大多数单片机应用系统中,上电和按键这两种复位方法都应该提供(至少提供上电复位电路)。当然系统一般都不会单独提供两个复位电路,通常都是将这两个电路整合在一起,如图 2.5(b)所示。

在图 2.5 所示的复位电路中,干扰容易串入复位端,虽然在大多数情况下不会造成单片机的错误复位,但可能会引起内部某些寄存器的错误复位。这时可在 RST 引脚上接一个去耦电容。

另外在有些单片机应用系统中的外围芯片也需要复位,如果这些复位端的复位电平要求和单片机的复位要求一致,则可以直接与之相连。常将 RC 电路接施密特电路后再接入单片机的复位端。这样系统可以有多个复位端,可以保证外部芯片和单片机可靠地同步复位。

3. 复位后的状态

MCS-51 单片机在复位期间不产生 ALE 和$\overline{\text{PSEN}}$信号,表明复位期间 CPU 不执行任何指令(做准备工作)。

复位完成后,PC 值为 0000H,表明复位后 CPU 自动从 0000H 处开始执行程序,这是程序最初运行的固定入口地址。而实际应用系统中的主程序不一定是(一般来说不是)从 0000H 开始存放。这样就要在程序存储器中最开始的地方放一条无条件转移指令,以便复位后 CPU 从 0000H 处开始执行这条转移指令,转移到主程序的实际位置。

如果是上电复位,所有内部数据 RAM 中的单元都是不确定的。如果是按键复位,因为未掉电,则所有内部数据 RAM 中的单元内容和复位前相同。复位后程序从头开始执行,因此在主程序中必须要有给相关单元进行初始化的程序段。

大多数 SFR 被清 0,特殊的有:

① P0～P3 都输出高电平,为这些 I/O 线的输入做好准备(详见 2.4 节)。

② SP 被初始化为 07H,即堆栈将从 08H 开始。在介绍堆栈指针时提到,一般将堆栈设置在 30H 以上为好。这是因为 00H～1FH 单元分属于工作寄存器区 0～3,20H～2FH 单元又为内部 RAM 中的位空间,这些单元和位有可能被使用。将堆栈设置在 30H 以上就是避免在进行堆栈操作时对这些单元内容的破坏。

那为什么复位后不直接初始化 SP 在 30H 以上呢?虽然 00H～1FH、20H～2FH 空间在程序中可能被使用(当然并不一定都被使用到),但其中又以 00H～07H 这组寄存器最为常用。实际上很少有程序不使用到工作寄存器,而且在相对简单的程序中一组工作寄存器也就够用了。内部 RAM 的空间很有限,每一个单元都很宝贵,所以就将 SP 初始化为 07H,使堆栈默认是从 08H 开始,主要就是为了避开 00H～07H 这一组工作寄存器。

当然,如果程序中要使用 08H～2FH 中的这部分空间,就应将 SP 设置为 30H 或更大。因此,程序开头处一般都会有一条设置堆栈指针的指令。

2.4 MCS－51 单片机的并行 I/O 口

MCS－51 单片机在 I/O 接口方面很有特点，主要体现在接口都做在片内，接口设计工作就不需要用户来做。这样也提高了单片机与外设数据交换的处理速度。在一些单片机应用系统中，可以直接将一些芯片接在单片机的接口上，而不需要增加任何电路。

MCS－51 单片机有 4 个并行 I/O 口，分别是 P0、P1、P2 和 P3 口。

2.4.1 并行 I/O 口的基本结构

P0、P1、P2 和 P3 口的位结构如图 2.6、图 2.7、图 2.8、图 2.9 所示。这 4 个端口的基本结构很相似，都有锁存器、输出驱动器和输入缓冲器等部件。P0～P3 实际上只是其中的锁存器，属于 SFR。为了方便，将所有这些部件笼统地称为 P0～P3 口。

注意： 每个端口中都有 2 个三态缓冲器用于读操作。

图 2.6～图 2.9 中下面的缓冲器用于直接读端口引脚处的数据。当执行由端口输入的指令（如“MOV A，P1”）时，此缓冲器打开，端口上的数据传送至内部总线。上面的缓冲器用于读锁存器 Q 端的数据，以适应“读—修改—写”类指令。这类指令都是先读锁存器，随之可能对读入的数据进行修改，然后再写回锁存器（如“ORL P0，A”指令）。对于这类指令，不直接读引脚而读锁存器是为了避免可能的读错。

2.4.2 P0 口的结构

P0 口的位结构如图 2.6 所示。P0 口可作一般的 I/O 口使用。当外接存储器时，分时用于地址和数据。

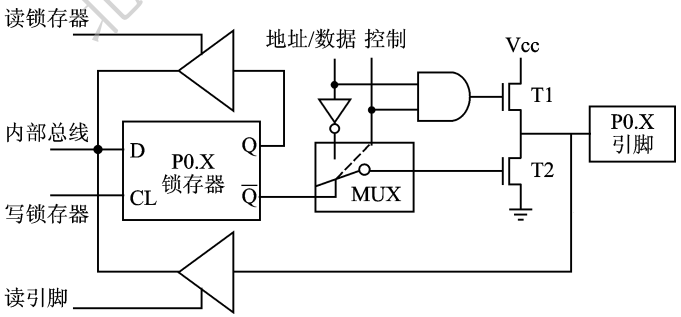


图 2.6 P0 口的位结构

在 4 个并行口中，只有 P0 口是分时复用的。如果扩展了外部存储器或无锁存功能的外部接口芯片，则要在它们和 P0 口之间加一个地址锁存器。

P0 口在结构上包含 1 个输出锁存器、2 个三态输入缓冲器、1 个输出驱动电路和 1 个输出控制电路。

1. 输出控制电路

由于 P0 口是分时使用的多功能口，在某时刻工作于什么状态需要相关信号的控制。输出控制电路由 1 个“与”门、1 个反相器和多路转换开关 MUX 组成。

当 P0 口用作普通的 I/O 时，控制信号为 0，“与”门输出为 0，T1 截止，MUX 下通，T2 起作用。CPU 将需要输出的数据经内部总线，送到 D 锁存器，锁存数据到 Q、$\overline{Q}$端，经过 MUX、T2，反相后送到引脚。

当 P0 口用于数据/地址总线时，控制信号为 1，“与”门输出取决于数据/地址信息，MUX 上通。此时 P0 的工作与图 2.6 左半部分的电路结构无关。

当 P0 口作为 I/O 口使用时，具有锁存功能。当 P0 口用于数据/地址总线时，无锁存功能。

2. 输出驱动电路

输出驱动电路由一对场效应管 FET 组成，其工作状态受输出控制电路的控制。当 P0 口用于 I/O 口输出数据时，由于 T1 已经截止，因此必须外接上拉电阻(2～10 kΩ)才能有高电平输出。当用作地址/数据总线时输出数据时，不需要外接上拉电阻。

如果 P0 口用于 I/O 口输入时，必须先向电路中的锁存器写“1”。也就是说在读引脚时，应先对引脚初始化。

假设在读 P0.0 引脚之前已经执行了“CLR P0.0”指令，此时下面的场效应管 T2 通。假设此时读引脚 P0.0：若外部电路连接到 P0.0 的信号为 0，则读入的数据为 0，正确；若外部电路连接到 P0.0 的信号为 1，但由于 T2 通，导致读入的数据仍然为 0，错误。不仅读入的数据错误，而且对外部电路、单片机等部件都有影响，可能造成器件损坏。

因此在将端口作为输入口使用时，应先执行类似“MOV P0，#0FFH”的指令，使输出级的 2 个 FET 都截止，端口处于“悬浮”状态，用于高阻抗输入数据。

当然，如果在复位后的整个程序执行过程中，P0 口都是作为输入口的，这个过程可以省略。但如果在程序中有时将 P0 口用作输入口，有时将 P0 口用作输出口，则在每次输入时，都要进行相关的初始化。

不仅是 P0 口，P1、P2、P3 口在用作 I/O 输入时都要这样处理。这就是为什么单片机复位后 P0～P3 的状态是 FFH 的原因，其目的是为使这些端口的输入先做好准备。

2.4.3 P1 口的结构

P1 口是一个标准的准双向口。P1 口的位结构如图 2.7 所示。

在结构上，因 P1 口一般只用作普通的 I/O，故内部不需要模拟转换开关。

另外在输出驱动部分，输出驱动电路由内部上拉电阻与场效应管共同组成，因此 P1 口作为输出口使用时，无须再加上拉电阻。

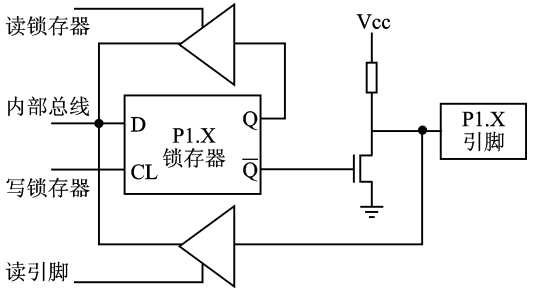


图 2.7　P1 口的位结构

2.4.4　P2 口的结构

P2 口也是双功能的，其位结构如图 2.8 所示。

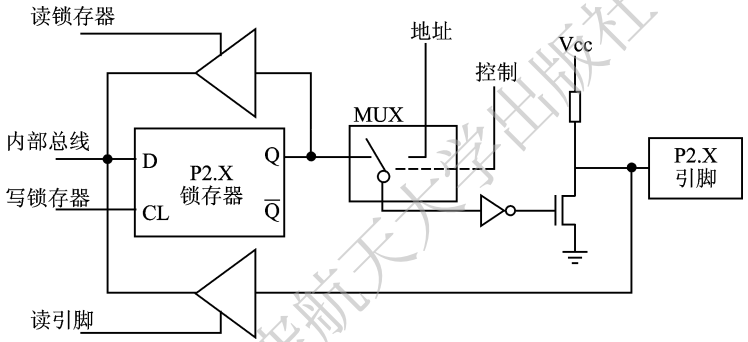


图 2.8　P2 口的位结构

与 P0 口相似，在结构上 P2 口也需要一个输出控制部分。

当系统中接有外部存储器时，P2 口要用于输出高 8 位地址。由于 P2 口不是分时使用的，P2 口本身又具备锁存功能，因此不需要加地址锁存器。

P2 口的输出驱动电路同 P1 口，在作为普通 I/O 口输出时，也无须再加上拉电阻。

2.4.5　P3 口的结构

P3 口的位结构如图 2.9 所示。

P3 口也是一个双功能口，当作为通用 I/O 口使用时，工作原理与 P1 和 P2 口类似。除用作通用 I/O 口外，P3 口还可工作于专用功能。

由于 P3 口的第二功能中有输出的，也有输入的，因此在输入方面多了一个缓冲器，用于第二功能的输入。

P3 口的输出驱动电路也同 P1 口，作为普通 I/O 口输出时，也无须再加上拉电阻。

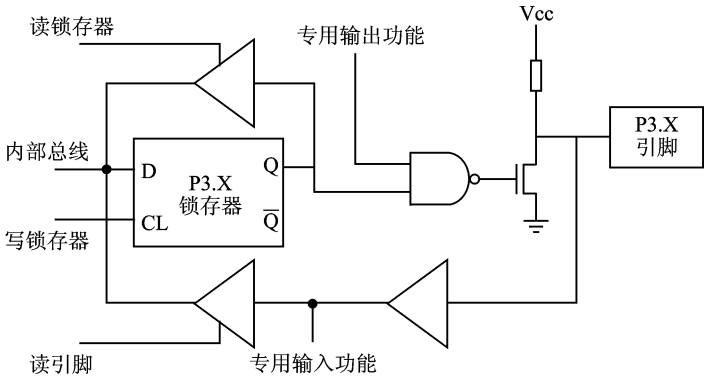


图 2.9　P3 口的位结构